Generative Active Learning for Long-tailed Instance Segmentation

Muzhi Zhu [1 *] Chengxiang Fan [1 *] Hao Chen [1] Yang Liu [1] Weian Mao [2 1] Xiaogang Xu [3 1] Chunhua Shen [1 4]

Abstract

Recently, large-scale language-image generative models have gained widespread attention and many works have utilized generated data from these models to further enhance the performance of perception tasks. However, not all generated data can positively impact downstream models, and these methods do not thoroughly explore how to better select and utilize generated data. On the other hand, there is still a lack of research oriented towards active learning on generated data. In this paper, we explore how to perform active learning specifically for generated data in the long-tailed instance segmentation task. Subsequently, we propose BSGAL, a new algorithm that online estimates the contribution of the generated data based on gradient cache. BSGAL can handle unlimited generated data and complex downstream segmentation tasks effectively. Experiments show that BSGAL outperforms the baseline approach and effectually improves the performance of long-tailed segmentation.

1. Introduction

Data is one of the driving forces behind the development of artificial intelligence. In the past, securing high-quality data was a time-consuming and laborious task. Yet, a large amount of high-quality data is crucial for a model to achieve breakthrough performance. Therefore, many active learning methods have emerged to explore the most informative samples from massive unlabeled data to achieve better model performance with minimal annotation costs. Currently, the rapid development of generative models has made it possible to obtain massive amounts of high-quality data, including long-tailed data, at a relatively low cost. In the field of visual perception, there have been many works utilizing generated data to improve perception tasks, including classification (Azizi et al., 2023), detection (Chen et al., 2023), and segmentation (Wu et al., 2023b). However, they often directly use generated samples as mixed training data (Yang et al., 2023) or as data augmentation (Zhao et al., 2023) without exploring how to better filter and utilize the data for downstream models.

[*]Equal contribution [1]Zhejiang University, China [2]The University of Adelaide, Australia [3]The Chinese University of Hong Kong, China [4]Ant Group. Correspondence to: Hao Chen <haochen.cad@zju.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

On the other hand, existing data mining and filtering methods, such as active learning, have only been validated on real data and are not suitable for generated data, as there are differences between generated data and real data regarding characteristics and usage scenarios. The differences are mainly as follows: 1) Existing data analysis methods are aimed at a limited data pool, while the scale of generated data is almost infinite. 2) Active learning is often carried out under a specified annotation budget. Thanks to the development of conditional generative models, the annotation cost of generated data can be almost negligible. However, this results in an unclear and uncertain quality of annotation in generated data compared to expert annotations. 3) There are differences in the distribution between real data and generated data, while the target data of previous methods do not have obvious distribution differences.

In response to the aforementioned challenges, we propose a novel problem called "Generative Active Learning for Long-tailed Instance Segmentation" (see Figure 1), which investigates how to utilize generated data effectively for downstream tasks. We focus on long-tailed instance segmentation for three main reasons. First, data collection for long-tailed categories is exceedingly arduous, and how to do classification well is currently a focus in the segmentation field (Kirillov et al., 2023; Li et al., 2023a; Yuan et al., 2024; **?**; Zou et al., 2023; Liu et al., 2023). Given that generated data has demonstrated the potential to alleviate this difficulty (Zhao et al., 2023; Xie et al., 2023; Fan et al., 2024), it is necessary to introduce generated data for long-tail segmentation tasks. Second, the quality requirements for generated data in long-tailed instance segmentation tasks are very high, and not all generated data can have a promoting effect. Therefore, it is necessary to further explore how to screen generated data. Third, this task itself is very comprehensive and challenging, which has more guiding significance for migration to real-world scenarios.

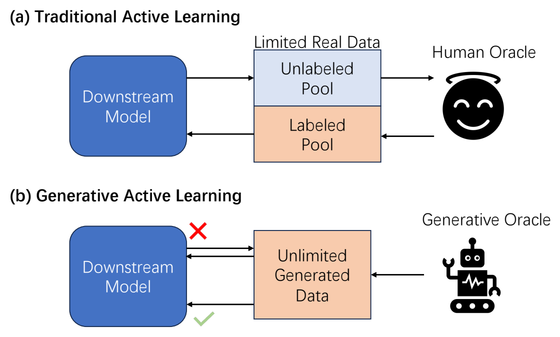

Figure 1. Comparison between Traditional Active Learning and Generative Active Learning frameworks. (a) Traditional Active Learning relies on a human oracle, therefore the annotation is accurate but with a limited budget, so the model is required to select the most informative unlabeled data. (b) Generative Active Learning, which relies on a generative oracle, has an unlimited labeled pool. However, the quality of annotation varies greatly, so the model must judiciously accept data.

Here, our objective is to design a generative active learning pipeline targeted at long-tailed image segmentation tasks. Inspired by data-influence analysis methods (Ling, 1984; Koh and Liang, 2017), we first use the change of the loss value to provide an estimation of the contribution of a single generated instance in the ideal case, as discussed in Section 4.1. Employing the first-order Taylor expansion, we introduce an approximate contribution estimation function based on the gradient dot product, which avoids repeated calculations on the test set in the offline setting. Based on this technique, we conduct a toy experiment on CIFAR-10 (Krizhevsky et al., 2009) in Section 5.1.1 along with a qualitative analysis of each sample on the LVIS dataset in Section 5.2.2, which preliminarily verify the feasibility of our approach. Subsequently, in Section 4.2 we explore how to apply this evaluation function to the actual segmentation training process. We propose the Batched Streaming Generative Active Learning algorithm (BSGAL), which allows for online acceptance or rejection of each batch of generated data. Additionally, based on the first-order gradient approximation, we maintain a gradient cache based on momentum updates to enable a more stable contribution estimation. Finally, experiments are carried out on the LVIS dataset, establishing that our method outperforms both unfiltered or CLIP-filtered counterparts under various backbones. Notably, in the long-tailed category, there is an over 10% improvement in AP_r (Gupta et al., 2019). In addition, We conduct a series of ablation experiments to delve into the particulars of our algorithmic design, including the choice of loss, the way of contribution estimation, and the sampling strategy of the test set. To summarize, our primary contributions are detailed below:

- We introduce a novel problem called "Generative Active Learning for Long-tailed Instance Segmentation": how to design an effective method focused on the successful using generated data, aimed at enhancing the performance of downstream segmentation tasks. Existing data analysis methods are neither directly applicable to generated data nor have they been affirmed as efficient for such data.

- We propose a batched streaming generative active learning method (BSGAL) based on gradient cache to estimate generated data contribution. This pipeline can adapt to the actual batched segmentation training process, handle unlimited generative data online, and effectively enhance the performance of the model.

- We carry out experiments on the LVIS dataset and demonstrate that our method outperforms both unfiltered and CLIP-filtered methods. Our model surpasses the baseline by +1.2 on AP_{box} and +3.62 on AP_r. We also conduct more comprehensive analysis experiments on the design or hyperparameters of our method.

2. Related work

2.1. Generative Data Augmentation

Generative data augmentation (GDA) refers to using generative models to synthesize additional data for augmentation. With the continuous improvement in the capabilities of generative models (Goodfellow et al., 2020; Saharia et al., 2022; Rombach et al., 2022b), GDA has become a popular technique for improving model performance. Several works use GDA in perceptual tasks such as classification (Feng et al., 2023; Zhang et al., 2023; Azizi et al., 2023), detection (Zhao et al., 2023; Chen et al., 2023), and segmentation (Li et al., 2023b; Wu et al., 2023b;a; Xie et al., 2023). Early works (Zhang et al., 2021; Li et al., 2022) involve the use of generative models like Generative Adversarial Networks (GANs) (Goodfellow et al., 2020) to generate additional training data. With the evolution of diffusion models, recent works (Azizi et al., 2023; Li et al., 2023b; Wu et al., 2023b; Yang et al., 2023; Zhao et al., 2023) favor using high-quality diffusion models such as Imagen (Saharia et al., 2022) and Stable Diffusion (Rombach et al., 2022b) for data generation. X-Paste (Zhao et al., 2023) has proven the strategy of using copy-paste to be more effective than directly using generated data for mixed training, and for the first time demonstrated that using generated data can enhance the performance of segmentation models on the long-tailed segmentation dataset LVIS (Gupta et al., 2019). Therefore, we consider it as the baseline for our work. However, while previous works have investigated the effects of GDA on different tasks, there has been limited exploration on how to better filter and utilize generative data for downstream perception models.

2.2. Active Learning and Data Analysis

Analysis of the information or contribution of data samples to a model has been extensively studied long before the advent of deep learning. Among them, two fields are most relevant to our work, one is active learning, and the other is training data influence analysis.

Active learning (Ren et al., 2021) mainly focuses on how to explore the most informative samples from massive unlabeled data to achieve better model performance with minimal annotation costs. Generally speaking, active learning can be divided into two categories. One is uncertainty-based active learning, which measures the uncertainty of samples by the posterior probability of the predicted category (Lewis and Catlett, 1994; Lewis, 1995; Goudjil et al., 2018) or the entropy of the predicted distribution (Joshi et al., 2009; Luo et al., 2013), and then selects the most uncertain samples for annotation. The other is diversity-based active learning, which is based on clustering (Nguyen and Smeulders, 2004) or core-set (Sener and Savarese, 2018) methods. They attempt to mine the most representative samples from the data to achieve minimal annotation costs. Recently, active learning in deep learning also tends to adopt a batch-based sample querying method (Ash et al., 2020), which is consistent with our work. The most relevant work to our work is VeSSAL (Saran et al., 2023), which does batched active learning in a streaming setting and samples in a gradient space. Another relatively related work (Mahapatra et al., 2018) trains a GAN on medical images, using the GAN to generate more data for active learning.

Training data influence analysis (Hammoudeh and Lowd, 2022) explores the relationship between training data samples and model performance, which can be divided into retraining-based (Ling, 1984; Roth, 1988; Feldman and Zhang, 2020) and gradient-based (Koh and Liang, 2017; Yeh et al., 2018). The most typical retraining-based method is Leave-One-Out (Ling, 1984; Jia et al., 2021), which measures the contribution of a sample to the model by removing a sample from the training set and then retraining the model. However, this method is obviously impractical for modern large-scale datasets. Therefore, many gradient-based methods have emerged recently, which use gradients to approximate the change of loss, such as using first-order Taylor expansion or Hessian matrix, to estimate the influence of samples. The most relevant work to ours is TracIn (Pruthi et al., 2020), which implements heuristic dynamic estimation through first-order gradient approximation and stored checkpoints. Unlike our work, the ultimate goal of TracIn is to estimate and filter out mislabeled samples in the training set through self-influence. Moreover, TracIn is only applicable to small-scale classification datasets, it is difficult to migrate to larger and complex tasks like segmentation, let alone handle nearly infinite generated data. Our work succeeds in designing an automated pipeline for utilizing generated data to enhance downstream perception tasks.

Algorithm 1 Pipeline for copy-paste baseline

```
Require: labeled real data $\mathcal{R}$, generated data $\mathcal{G}$, batch size $B$, number of iterations $T$, pretrained segmentation network $f$ with parameters $\theta$, maximum number of paste instances $K$ for each image
1: for $t = 1$ to $T$ do
2:     Sample a batch of real data $\mathcal{R}_b \in \mathcal{R}$
3:     for $\mathbf{I}_r \in \mathcal{R}_b$ do
4:         Get a random number $k$ from $[0, K]$
5:         Sample $k$ instances from $\mathcal{G}$ in a class-balanced way to get $\widehat{\mathbf{I}}_r, \widehat{\mathbf{Y}}_r = Copypaste(\mathcal{G}_b, \mathbf{I}_r)$
6:     end for
7:     Train $f$ on this augmented data $\widehat{\mathcal{R}}_b$ and update $\theta$
8: end for
9: return: Final segmentation network $f$ with parameters $\theta$
```

Most importantly, the above work is all done on relatively simple classification tasks, and only a few works have explored more complex perception tasks such as detection (Shrivastava et al., 2016; Liu et al., 2021) and segmentation (Jain and Grauman, 2016; Vezhnevets et al., 2012; Casanova et al., 2020), but they are all aimed at real data. Our work is the first to explore the generated data on the complex perception task of long-tail instance segmentation.

3. Preliminary

Formally, in generated data augmented instance segmentation tasks, we have a set of labeled real data $\mathcal{R} = \{(\mathbf{I}_r, \mathbf{Y}_r)\}$ and a set of generated data (with noisy label) $\mathcal{G} = \{(\mathbf{I}_g, \mathbf{Y}_g)\}$ where $\mathbf{I}$ is the image and $\mathbf{Y}$ is the label for instance segmentation. Our goal is to effectively utilize the existing data $\mathcal{R}$ and $\mathcal{G}$ to train a segmentation network f parameterized by θ to achieve optimal performance on unseen test data $\mathcal{U} = \{(\mathbf{I}_u, \mathbf{Y}_u)\}$, that is, minimize the loss on the test set $L_{\mathcal{U}}(\theta) = \sum_{(\mathbf{I}_u, \mathbf{Y}_u) \in \mathcal{U}} \ell(\mathbf{I}_u, \mathbf{Y}_u; \theta)$, where ℓ represents the loss of the segmentation network. The most direct way to utilize the two types of data is to perform joint training, and X-Paste proves that using the copy-paste (Ghiasi et al., 2021) method to add instances from $\mathbf{I}_g$ to real images $\mathbf{I}_r$ can achieve better results. Although our method is universal, we build it upon a copy-paste baseline. For the convenience of subsequent description, we record this operation as $Copypaste$ such that $\widehat{\mathbf{I}}_r, \widehat{\mathbf{Y}}_r = Copypaste(\mathcal{G}_b, \mathbf{I}_r)$, where $\mathcal{G}_b \in \mathcal{G}$ is a subset sampled from $\mathcal{G}$, $\widehat{\mathbf{I}}_r$ and $\widehat{\mathbf{Y}}_r$ represent the image and the label obtained after pasting the instance in $\mathcal{G}_b$ to $\mathbf{I}_r$.

As shown in Algorithm 1, it displays the overall process of our baseline, which does not consider the different impacts each sample could impose on the model. In other words, our aim is to identify a function, $\phi(g, \theta)$, capable of gauging the contribution of any given generated sample $g \in \mathcal{G}$ to the current model f. Then, via this scoring mechanism, we can filter and retain the most helpful samples for the model and simultaneously discard those that are useless or even harmful to the model.

4. Our method

4.1. Estimation of Contribution in the Ideal Scenario

Here we first provide the ideal estimation of the contribution of an independent sample. Assuming that a target test set $\mathcal{U}$ is given, the contribution of $g \in \mathcal{G}$ to f can be measured by calculating the change in the loss of f on $\mathcal{U}$, that is,

$$\phi(g, \theta) = L_{\mathcal{U}}(\theta) - L_{\mathcal{U}}(\theta + \Delta\theta_g) \tag{1}$$

where $\Delta\theta_g$ represents the one-step gradient update on g, that is,

$$\Delta\theta_g = -\alpha \nabla_\theta \ell(\mathbf{I}_g, \mathbf{Y}_g; \theta) \tag{2}$$

where α represents the learning rate.

Moreover, we can employ the classic first-order Taylor expansion to approximate $\phi(g, \theta)$, which has also been widely used in previous work (Pruthi et al., 2020; He et al., 2023). Note that, it is possible to use more sophisticated methods here, *e.g.*, Koh and Liang (2017).

Lemma 4.1. *The loss of a network f on a dataset $\mathcal{U}$ can be approximated by a first-order approximation:*

$$L_{\mathcal{U}}(\theta + \Delta\theta) = L_{\mathcal{U}}(\theta) + \Delta\theta^T \nabla_\theta L_{\mathcal{U}}(\theta) + O(\Delta\theta^2) \tag{3}$$

So the contribution of g to f can be approximated by:

$$\phi(g, \theta) \approx \phi_a(g, \theta) = \alpha \nabla_\theta \ell(\mathbf{I}_g, \mathbf{Y}_g; \theta)^T \nabla_\theta L_{\mathcal{U}}(\theta) \tag{4}$$

Remark 4.2. **Advantages of first-order approximation:** In the case of a fixed test set $\mathcal{U}$, it eliminates the need for multiple forward computations on $\mathcal{U}$, requiring only one forward computation and one backward propagation. so it brings the possibility expand the scale of $\mathcal{U}$. Besides, the fact that there is no need to update the weights makes our subsequent designs more flexible.

Remark 4.3. **(Test set)** In real-world scenarios, the test set $\mathcal{U}$ is typically unknown. A straightforward solution might be to reserve a part of the training set $\mathcal{R}$ as an unseen test set. However, this approach poses issues for datasets with significant category imbalance, such as LVIS, where certain long-tailed categories only appear once in the training set. In light of this, we resort to a strategy that involves sampling a subset of $\mathcal{R}$ to serve as the test set $\mathcal{U}$ for each iteration. See Section 5.2.2 for more details.

4.2. Batched Streaming Generative Active Learning

While we have proposed an ideal estimation of contribution in the preceding section, it is not directly transferable to a real segmentation training process. The specific reasons are as follows:

1. In the previous discussion, we posit that $g \in \mathcal{G}$ is an independent sample, yet we actually paste multiple instances from $\mathcal{G}$ to I_r. This results in reciprocal influences among these instances, as well as interactions with I_r.

2. We conduct batch data training, eliminating the possibility of estimating each instance individually, as this would provoke excessive computation.

3. Our model undergoes constant updates. Therefore, even the same sample's contribution to the model varies under different training stages. Furthermore, given the near-infinite data pool, the entire training process closely resembles a streaming process(Saran et al., 2023). After each data entry, we must decide whether to include this data in the present update.

For the first and second points, we redefine the contribution function $\phi(g, \theta)$ so that it can simultaneously consider the mutual influence between multiple instances and the real image I_r, and can also adapt to batch data training.

Definition 4.4. Specifically, for a certain iteration t, we introduce a batch of data $\mathcal{R}_b$ and the corresponding instances $\mathcal{G}_b$ to be pasted. We will paste $\mathcal{G}_b$ to each image in $\mathcal{R}_b$ in a predefined random way to get $\widehat{\mathcal{R}}_b = \mathcal{R}_b \oplus \mathcal{G}_b$. Then our contribution function $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta)$ can be defined as

$$\phi(\mathcal{G}_b, \mathcal{R}_b, \theta) = L_U(\theta + \Delta\theta_{\mathcal{R}_b}) - L_U(\theta + \Delta\theta_{\widehat{\mathcal{R}}_b}) \tag{5}$$

where $\Delta\theta_{\widehat{\mathcal{R}}_b}$ represents a one-step gradient update on $\widehat{\mathcal{R}}_b$:

$$\Delta\theta_{\widehat{\mathcal{R}}_b} = -\alpha \nabla_\theta L_{\widehat{\mathcal{R}}_b}(\theta) \tag{6}$$

Likewise, $\Delta\theta_{\mathcal{R}_b}$ represents a one-step gradient update on $\mathcal{R}_b$.

In response to the third point, based on Definition 4.4, we can propose an algorithm called Batched Streaming Generative Active Learning (BSGAL), as shown in Algorithm 2.

The basic idea is to calculate the loss and gradient of the model on $\widehat{\mathcal{R}}_b$ and $\mathcal{R}_b$ respectively and use the two updated models to calculate the loss on $\mathcal{U}_b$. Then measure the contribution of $\mathcal{G}_b$ with the difference of two losses. Ultimately, we decide whether to accept this batch of generated data.

As pointed out in Remark 4.2, there are some advantages of using one-order approximation. Building upon Lemma 4.1,

Algorithm 2 Batched streaming generative active learning (BSGAL)

```
Require: labeled real data $\mathcal{R}$, generated data $\mathcal{G}$, batch size $B$, number of iterations $T$, pretrained segmentation network $f$ with parameters $\theta$, maximum number of paste instances $K$ for each image, one step learning rate $\alpha$, contribution threshold $\tau$
 1: for $t = 1$ to $T$ do
 2:   Sample a batch of real data $\mathcal{R}_b \in \mathcal{R}$ with batch size $B_{accept}$.
 3:   for $(\mathbf{I}_r, \mathbf{Y}_r) \in \mathcal{R}_b$ do
 4:     Get a random number $k$ from $[0, K]$
 5:     Sample $k$ instances in a class-balanced way from $\mathcal{G}$ to get $\widehat{\mathbf{I}}_r, \widehat{\mathbf{Y}}_r = Copypaste(\mathcal{G}_b, \mathbf{I}_r)$
 6:   end for
 7:   Merge all the augmented image and label pairs to get $\widehat{\mathcal{R}}_b = \mathcal{R}_b \oplus \mathcal{G}_b$
 8:   Calculate loss on $\mathcal{R}_b$ and $\widehat{\mathcal{R}}_b$ and get the gradient $\nabla_\theta L_{\mathcal{R}_b}(\theta)$ and $\nabla_\theta L_{\widehat{\mathcal{R}}_b}(\theta)$
 9:   Sample a batch of data $\mathcal{U}_b \in \mathcal{R}$ as test set with batch size $B_{test}$
10:   Calculate the contribution $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta)$ using Equation (5)
11:   if $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta) > \tau$ then
12:     Train $f$ on this augmented data $\widehat{\mathcal{R}}_b$ and update $\theta$
13:   else
14:     Train $f$ on this real data $\mathcal{R}_b$ and update $\theta$
15:   end if
16: end for
17: return: Final segmentation network $f$ with parameters $\theta$
```

we can further approximate $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta)$ as

$$\phi(\mathcal{G}_b, \mathcal{R}_b, \theta) \approx \alpha \nabla_\theta (L_{\widehat{\mathcal{R}}_b}(\theta) - L_{\mathcal{R}_b}(\theta))^T \nabla_\theta L_U(\theta) \quad (7)$$

So the Algorithm 2 can be further simplified by using Equation (7) in Line 10.

Remark 4.5. **(Batch size)** It is crucial to note that three distinct batch sizes here, one is the batch size B_{train} for model update, one is the batch size B_{accept} for calculating the data contribution to determine whether to accept, and the other is the batch size B_{test} of the test set formed by sampling. These three are not necessarily identical. Ideally, a smaller B_{accept} is preferable – when the batch size is reduced to 1, it allows for a more accurate per-image estimation. In the actual implementation process, we execute this algorithm independently on each GPU, except for summing up all losses when updating the model. Consequently, the batch size $B_{train} = B_{accept} \times \#GPU$.

Usually, due to the limitation of GPU memory, $\mathcal{U}_b$ can only take a very small batch size B_{test}, which will lead to the instability and inaccuracy of the estimation of $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta)$. Thanks to the one-order approximation, we can consider more test data when calculating $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta)$. Specifically, we can keep a grad cache to record the grad obtained on other batches in the previous iterations. Every time we get the current batch $\nabla_\theta L_{\mathcal{U}_b}(\theta)$, we will update the grad cache in a momentum way. Then when calculating $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta)$, we use the grad in the grad cache to replace the grad of the current batch, which is equivalent to expanding the scale of $\mathcal{U}_b$, thereby achieving a more stable estimation.

Algorithm 3 Contribution estimation based on gradient cache

```
Require: gradient $\nabla_\theta L_{\mathcal{R}_b}(\theta)$, gradient $\nabla_\theta L_{\widehat{\mathcal{R}}_b}(\theta)$, momentum coefficient $\beta$, current iteration $t$, model parameters $\theta$, sampled test set $\mathcal{U}_b$
1: Calculate loss on $\mathcal{U}_b$ to get $L_{\mathcal{U}_b}$ and get the grad $\nabla_\theta L_{\mathcal{U}_b}(\theta)$
2: if $t == 1$ then
3:   Initialize grad cache $\mathbf{C} = \nabla_\theta L_{\mathcal{U}_b}(\theta)$
4: else
5:   Update the grad cache with the grad of current batch $\mathbf{C} = \beta\mathbf{C} + (1-\beta)\nabla_\theta L_{\mathcal{U}_b}(\theta)$
6: end if
7: Calculate the contribution of $\mathcal{G}_b$ as $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta) = \alpha \nabla_\theta (L_{\widehat{\mathcal{R}}_b}(\theta) - L_{\mathcal{R}_b}(\theta))^T \mathbf{C}$
8: return: $\phi(\mathcal{G}_b, \mathcal{R}_b, \theta)$
```

However, this batch size B_{test} is not the bigger the better. Our subsequent experiments prove that if we use the grad cache to approximate the entire training set $\mathcal{R}$ as $\mathcal{U}_b$, it will lead to a decrease in the performance of the model. We believe that this is caused by overfitting. When we approximate the entire training set $\mathcal{R}$ as $\mathcal{U}_b$, we can only screen out samples similar to the training set $\mathcal{R}$, thereby inhibiting the diversity of the data. Diversity is also an important factor considered in many active learning works. We believe that the design here parallels the idea of diversity-based active learning (Sener and Savarese, 2018; Geifman and El-Yaniv, 2017) using a core-set, with the only difference being that core-set methods mine the most representative samples from unlabeled data, while we randomly sample some from the labeled real data to act as a kind of "core-set". In our task, the diversity of generated data is particularly crucial for long-tailed categories, which can effectively bridge the gap between scarce training data and real-world distribution. Therefore, we need to balance the stability of the estimation and the diversity of the data. Thus, we use the momentum method to update the grad cache, which can ensure the stability of the estimation to a certain extent, and at the same time will not inhibit the diversity of the data.

The modified contribution estimation algorithm for the final BSGAL is shown in Algorithm 3.

Remark 4.6. **(Extension to offline learning)** Offline learning needs to satisfy the following two assumptions: 1. The generated data is limited or small in scale. 2. The model parameters will not change significantly during the fine-tuning process. Our method can also be easily extended to the case of offline learning. Specifically, we only need to use a fixed model, and then use the entire dataset $\mathcal{R}$ as the test set $\mathcal{U}$. Forward and backward once on the entire dataset, we can get the gradient $\nabla_\theta L_{\mathcal{U}}(\theta)$ for the entire dataset. Then we can use this gradient to calculate the contribution of each generated sample. Using Equation (4), we can estimate the contribution of each generated sample.

5. Experiments

First, we perform some analytical experiments in an offline setting(as discussed in Remark 4.6) to verify the feasibility of our method and also to facilitate a better understanding of our method for readers. Then, we conduct the main experiments under the online setting, compared with our baseline. Key ablation studies are also conducted to substantiate the efficiency of our method. Detailed information about the implementation can be found in Appendix A.

5.1. Offline Setting

5.1.1. CIFAR-10

In this section, we conduct a toy experiment on CIFAR-10 (Krizhevsky et al., 2009) to verify our method. The original CIFAR-10 dataset comprises five splits within the training set, each containing 10,000 images, and a test set equally housing 10,000 images. We use the first split in the training set as our training set $\mathcal{R}$, and the remaining 4 splits are added with noise of different scales (40,100,200,400) to simulate the generated data $\mathcal{G}$. We use the model trained only on the first split to perform offline mining and then use 1000 images in the first split as test set $\mathcal{U}$. By estimating the contribution of each sample, we can draw the distribution of the contribution of samples on different splits.

As shown in Figure 2, it is observable that with the escalating scale of noise, the distribution of contributions progressively shifts to the left. This indicates that excessive noise tends to negatively impact the model. Note that the split with a noise of 0 is our training set, so we can see that the contribution values of these samples are concentrated around zero. In other words, these samples can no longer bring positive effects to the model because they have been fully utilized in previous training. This observation is consistent with some previous active learning work (Cai et al., 2013; Ash et al., 2021; Saran et al., 2023), where they also estimate the amount of information or the difficulty level of samples through gradients. However, they do not consider the positive or negative contributions but only select samples with larger absolute values. We further conduct quantitative experiments, as shown in Table 1, to prove that using our method to select data can effectively improve the performance of the model.

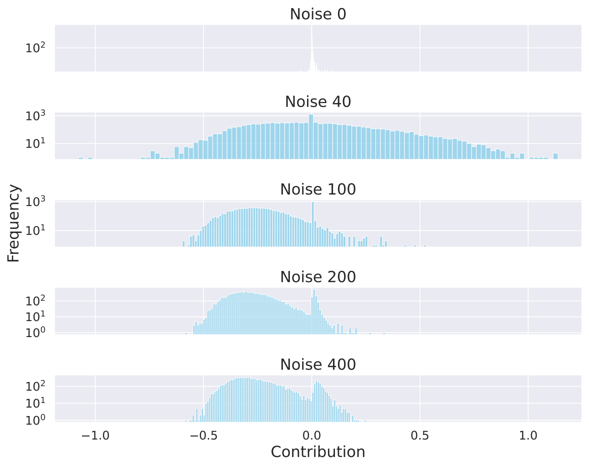

Figure 2. The distribution of contributions under different noise scales.

Table 1. Using our method to select samples brings improvement to the model.

Training set	Accuracy (avg $\pm$ std)
only $\mathcal{R}$	86.28 ± 0.55
$\mathcal{R}$ + all $\mathcal{G}$	86.71 ± 0.21
$\mathcal{R}$ + selected $\mathcal{G}$	87.61 ± 0.20

5.1.2. LVIS

We further carry out offline experiments on the generated data of LVIS categories, qualitatively examining the efficacy of our method. We calculate the gradient $\nabla_\theta L_{\mathcal{U}}(\theta)$ preemptively on the LVIS training set with a trained model.

This gradient then serves to estimate each instance's contribution. Subsequently, we rank these instances in decreasing order of their contribution, facilitating per-image analysis. As an illustrative example, we use a 'bun' category from the LVIS, because we discover that Stable Diffusion does not perform optimally within this category, often leading to confusion between 'bun' and 'bunny', thereby resulting in the generation of ambiguous data. As depicted in Figure 3, it can be observed that the instances having the most significant contributions are nearly unambiguous, whereas the instances with minimal contributions are mostly incorrect, resulting in rabbit images being generated. Therefore, through our method, we can effectively filter out the generated data with ambiguity.

To verify the indispensability of online learning, we first use the offline method to filter the generated data for training and compare it with our baseline. As shown in Figure 4, the offline method can only bring a slight improvement to the

Table 2. Main results on LVIS. "+CLIP" means using CLIP to filter the generated data.

Method	Backbone	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}	AP_c^{box}	AP_c^{mask}	AP_f^{box}	AP_f^{mask}
Baseline	R50	34.20	30.39	24.33	22.21	33.23	29.57	39.63	34.89
Baseline + CLIP	R50	34.35	30.70	25.99	24.38	32.83	29.41	39.71	34.92
Ours	R50	**35.40**	**31.56**	**27.95**	**25.43**	**34.14**	**30.56**	**40.07**	**35.37**
Baseline	Swin-L	49.57	43.85	44.87	39.66	49.74	44.64	51.46	44.82
Baseline + CLIP	Swin-L	49.80	44.51	45.28	40.62	49.33	44.96	**52.30**	**45.72**
Ours	Swin-L	**50.47**	**44.85**	**47.55**	**42.37**	**50.43**	**45.47**	51.79	45.26

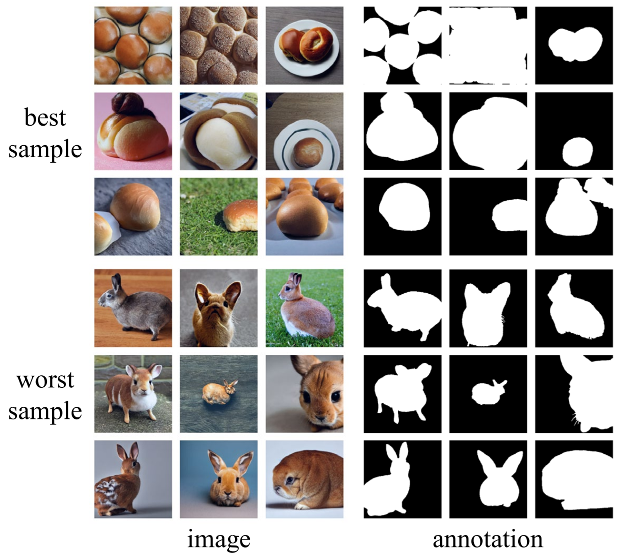

Figure 3. The best and worst samples found using our contribution estimation function for a LVIS class 'bun'.

final model performance. In addition, in the early stage of model training, this performance improvement is still quite obvious, but with the training process, this performance improvement gradually diminishes. We conjecture that this trend is likely due to the offline contribution estimation's reliance on the initial model, and as the model undergoes training, the parameters change significantly, which leads to the inaccuracy of the offline contribution estimation. Therefore, the necessity arises for online contribution estimation.

5.2. Online Setting

5.2.1. Main results

To validate the effectiveness of our method in handling long-tailed segmentation tasks, we perform experiments on the LVIS (Gupta et al., 2019) dataset. A strong baseline —— X-Paste (Zhao et al., 2023), is compared with our method. We further examine the impact of the usage (or non-usage) of CLIP (Radford et al., 2021), as mentioned in their paper, for filtering generated data. Given that X-Paste does not open source the generated data used, we have re-implemented the data generation pipeline and generated thousands of images

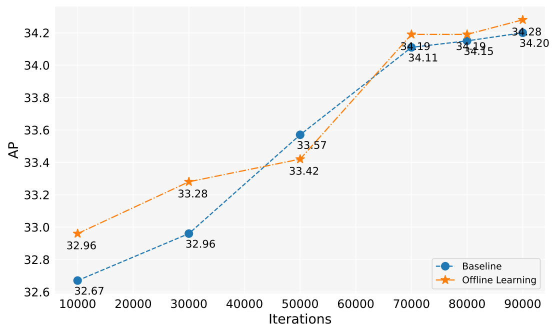

Figure 4. Performance of the model under different iterations.

for each category of LVIS. The specific generation details are shown in the appendix. In addition, original X-Paste also uses retrieval to obtain extra real data and uses Copy-Paste to augment the real training data. Since the primary focus of our methodology is on the selection of generated data, to mitigate the influences from those additional factors, we choose to refrain from using these tricks.

As shown in Table 2, following X-Paste, the segmentation architecture we used is CenterNet2 (Zhou et al., 2021), and we test two different backbones, ResNet50 (He et al., 2016) and Swin-L (Liu et al., 2022). It can be observed that the impact of filtration via CLIP is rather subtle, with the AP^{box} witnessing an increment of merely $0.1 \sim 0.2$ points. Furthermore, we also find that CLIP filtering shows a more significant improvement on AP^{mask}. Conversely, our method continues to deliver substantial improvements across all categories when contrasted with the use of CLIP filtration. This improvement is especially notable for long-tailed categories, AP_r^{box} increases by $2 \sim 2.3$, and AP_r^{mask} increases by $1 \sim 1.7$.

5.2.2. Ablation

Table 3. Comparison of different L_{test} on contribution estimation.

Loss	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}
cls	**35.24**	**31.49**	**28.14**	**25.74**
cls stage0	34.94	31.23	26.34	24.27
all	34.98	30.94	26.91	23.87

Table 4. Comparison of different algorithms we designed.

Method	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}	AP_c^{box}	AP_c^{mask}	AP_f^{box}	AP_f^{mask}
Loss estimate (Algorithm 2)	35.11	31.29	27.47	25.10	33.55	30.14	**40.20**	35.30
Grad estimate (Equation (7))	35.24	31.49	**28.14**	**25.74**	33.67	30.31	40.12	35.34
Grad cache (Algorithm 3)	**35.40**	**31.56**	27.95	25.43	**34.14**	**30.56**	40.07	**35.37**
Grad cache (global)	35.07	31.38	26.84	24.82	33.74	30.36	40.18	**35.40**

Loss. The algorithm in Section 4.2 deploys two distinct types of loss. First is the loss used to train and finally update the model, L_{train}, which is subject to the original network design. In our case, We utilize the loss in CenterNet2 (Zhou et al., 2021). For the instance segmentation task, the actual L_{train} is composed of multiple tasks and multiple stages of Loss. Specifically, it can be written in the following form[1]:

$$L_{train} = \sum_{i=1}^{S} (L_{cls}^i + L_{box}^i + L_{mask}^i) \tag{8}$$

where S represents the number of stages, L_{cls}^i, L_{box}^i, L_{mask}^i represent the classification loss, regression loss, and segmentation loss of the i-th stage, respectively.

Second is the test loss L_{test} used to estimate the contribution, which does not need to be consistent with L_{train}. Considering that the main purpose of introducing generated data is to improve classification on long-tailed categories, L_{cls} may play a dominant role. Thus, we compare the effects of different L_{test} on contribution estimation in Table 3.

In this experiment, we are based on Equation (7), where "cls" denotes using all stages of L_{cls} "cls stage0" denotes only using the first stage of L_{cls}, and "all" denotes using all Loss in L_{train}. The findings indicate that if only the L_{cls} of first stage is used, the efficiency of contribution estimation diminishes. Additionally, utilizing all losses within L_{train} does not surpass the performance of solely exploiting L_{cls}, which also verifies our assumption that L_{cls} dominates in contribution estimation, and other losses may cause interference. Therefore, our final decision is to rely singularly on L_{cls} for estimating contribution.

Contribution estimation. We are interested in whether the three algorithms proposed in Section 4.2 are effective. Therefore, we conduct comparative experiments here, and the specific results are presented in Table 4.

We observe that there's not a significant difference in performance among the three algorithms, all of which demonstrated efficacy when compared to the baseline. Overall, Algorithm 3 has the best effect, which is also the algorithm we finally adopted. Hence, our proposed method of using a larger test set $\mathcal{U}_b$, ensuring a smoother and more stable contribution estimation by updating the grad cache with momentum, is proven effective. Compared with Grad estimate and Loss estimate, it can be proved that the first-order approximation will not bring significant performance loss.

[1]In CenterNet2, there is only one stage for mask loss. In addition, there are some other losses, which are not listed for simplicity.

As discussed in Section 4.2, there exists a trade-off between the stability of the estimation and the diversity of the filtered data. To verify this, we additionally added the fourth experiment, where we use global average pooling to estimate the contribution. Specifically, We modify the original update method $\mathbf{C} = \beta\mathbf{C} + (1-\beta)\nabla_\theta L_{\mathcal{U}_b}(\theta)$ to be related to the current iteration t, $\mathbf{C} = \frac{t-1}{t}\mathbf{C} + \frac{1}{t}\nabla_\theta L_{\mathcal{U}_b}(\theta)$. That is, $\mathbf{C} = \frac{1}{t}\sum_{i=1}^{t} \nabla_{\theta_i} L_{\mathcal{U}_b^i}(\theta_i)$.

However, this modification incurs a slight performance drop. We believe that this is due to enforcing the generated data to align with the distribution of the entire training set and suppressing the diversity of the data. For long-tailed categories with relatively few real data, this diversity is quite significant. That's why using global average pooling, AP_r^{box} and AP_r^{mask} have a significant drop, while AP_f^{mask} exhibits an improvement. Correspondingly, Grad estimate, which solely uses the current batch to estimate the contribution of the test set, is most conducive to ensuring data diversity, so the performance of AP_r^{box} and AP_r^{mask} is the best.

Table 5. Comparison of different sampling strategies for $\mathcal{U}_b$.

Strategy	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}
all classes	35.21	31.38	26.75	24.21
pasted classes	**35.40**	**31.56**	**27.95**	**25.43**
all images	35.15	31.26	26.59	23.75

Sampling strategy. We compare three different sampling test set strategies: 1. Sample from all classes: Sample uniformly from all categories, and then sample uniformly from the image pool corresponding to the sampled category. 2. Sample from pasted classes: Sample uniformly from the categories in the generated data $\mathcal{G}_b$ used in this batch, and then sample uniformly from the image pool corresponding to the sampled category. 3. Sample from all images: Sample uniformly from all image pools.

As indicated in Table 5, it is evident that the approach of uniformly sampling from pasted categories delivers the most effective performance, thus we finalize on this sampling strategy. Especially in AP_r^{box} and AP_r^{mask}, the improvement of this sampling strategy is the most obvious. Compared with sampling from all images, sampling from class

Table 6. Comparison of random batch-level dropout and our method.

Method	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}
Random Dropout	34.73	30.96	25.05	22.69
Our method	**35.40**	**31.56**	**27.95**	**25.43**

can better ensure the balance of categories, thereby enhancing the impact on rare categories. As for the comparison with sampling from all classes, sampling from pasted classes is more directional, so the estimation of the contribution is more accurate, leading to a boost in overall performance.

Random batch-level dropout. Our algorithm is essentially to accept or reject the generated data on a batch-by-batch basis. Therefore, when the contribution evaluation of the data is completely invalid, our algorithm degenerates into a random batch-level dropout. To verify that it is not this random dropout that brings performance improvement, we conduct a random batch-level Dropout experiment with the same acceptance rate. Table 6 shows that although random dropout will also bring a slight improvement, compared with our method, there is still a very obvious gap, which shows that the improvement brought by our method is not entirely due to random dropout.

6. Conclusion

In this paper, we propose a new problem, how to design an effective method to realize the effective screening and utilization of generated data, to further improve the performance of downstream perception tasks. To address this problem, we propose a gradient-based generated data contribution estimation method and embed it into the actual training process. We design a complete pipeline that can automatically generate data to improve the performance of downstream perception tasks. Experiments prove that our method can achieve better performance than unfiltered or CLIP-filtered methods on long-tailed segmentation tasks.

Impact Statement

Our goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

This work was supported by the National Key R&D Program of China (No. 2022ZD0118700).

References

Jordan Ash, Surbhi Goel, Akshay Krishnamurthy, and Sham Kakade. Gone fishing: Neural active learning with fisher embeddings. In *Adv. Neural Inform. Process. Syst.*, pages 8927–8939, 2021.

Jordan T. Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In *Int. Conf. Learn. Represent.*, 2020.

Shekoofeh Azizi, Simon Kornblith, Chitwan Saharia, Mohammad Norouzi, and David J. Fleet. Synthetic data from diffusion models improves imagenet classification. *Transactions on Machine Learning Research*, 2023.

Wenbin Cai, Ya Zhang, and Jun Zhou. Maximizing expected model change for active learning in regression. In *IEEE Int. Conf. Data Mining*, pages 51–60. IEEE, 2013.

Arantxa Casanova, Pedro O. Pinheiro, Negar Rostamzadeh, and Christopher J. Pal. Reinforced active learning for image segmentation. In *Int. Conf. Learn. Represent.*, 2020.

Kai Chen, Enze Xie, Zhe Chen, Lanqing Hong, Zhenguo Li, and Dit-Yan Yeung. Integrating geometric control into text-to-image diffusion models for high-quality detection data generation via text prompt. *arXiv preprint arXiv:2306.04607*, 2023.

Chengxiang Fan, Muzhi Zhu, Hao Chen, Yang Liu, Weijia Wu, Huaqi Zhang, and Chunhua Shen. Divergen: Improving instance segmentation by learning wider data distribution with more diverse generative data. *arXiv preprint arXiv:2405.10185*, 2024.

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *Adv. Neural Inform. Process. Syst.*, 33: 2881–2891, 2020.

Chun-Mei Feng, Kai Yu, Yong Liu, Salman Khan, and Wangmeng Zuo. Diverse data augmentation with diffusions for effective test-time prompt tuning. In *Int. Conf. Comput. Vis.*, pages 2704–2714, 2023.

Yonatan Geifman and Ran El-Yaniv. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017.

Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2918–2928, 2021.

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.

Mohamed Goudjil, Mouloud Koudil, Mouldi Bedda, and Noureddine Ghoggali. A novel active learning method using svm for text classification. *Int. J. Autom. Comput.*, 15:290–298, 2018.

Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5356–5364, 2019.

Zayd Hammoudeh and Daniel Lowd. Training data influence analysis and estimation: A survey. *arXiv preprint arXiv:2212.04612*, 2022.

Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Sensitivity-aware visual parameter-efficient fine-tuning. In *Int. Conf. Comput. Vis.*, pages 11825–11835, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 770–778, 2016.

Suyog Dutt Jain and Kristen Grauman. Active image segmentation propagation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2864–2873, 2016.

Ruoxi Jia, Fan Wu, Xuehui Sun, Jiacen Xu, David Dao, Bhavya Kailkhura, Ce Zhang, Bo Li, and Dawn Song. Scalability vs. utility: Do we have to sacrifice one for the other in data importance quantification? In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8239–8247, 2021.

Ajay J Joshi, Fatih Porikli, and Nikolaos Papanikolopoulos. Multi-class active learning for image classification. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2372–2379. IEEE, 2009.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. In *Int. Conf. Comput. Vis.*, pages 4015–4026, 2023.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proc. Int. Conf. Mach. Learn.*, pages 1885–1894. PMLR, 2017.

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

David D Lewis. A sequential algorithm for training text classifiers: Corrigendum and additional data. In *Acm Sigir Forum*, pages 13–19. ACM New York, NY, USA, 1995.

David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine learning proceedings 1994*, pages 148–156. Elsevier, 1994.

Daiqing Li, Huan Ling, Seung Wook Kim, Karsten Kreis, Sanja Fidler, and Antonio Torralba. Bigdatasetgan: Synthesizing imagenet with pixel-wise annotations. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 21330–21340, 2022.

Feng Li, Hao Zhang, Peize Sun, Xueyan Zou, Shilong Liu, Jianwei Yang, Chunyuan Li, Lei Zhang, and Jianfeng Gao. Semantic-sam: Segment and recognize anything at any granularity. *arXiv preprint arXiv:2307.04767*, 2023a.

Ziyi Li, Qinye Zhou, Xiaoyun Zhang, Ya Zhang, Yanfeng Wang, and Weidi Xie. Open-vocabulary object segmentation with diffusion models. In *Int. Conf. Comput. Vis.*, pages 7667–7676, 2023b.

Robert F Ling. Residuals and influence in regression, 1984.

Yang Liu, Muzhi Zhu, Hengtao Li, Hao Chen, Xinlong Wang, and Chunhua Shen. Matcher: Segment anything with one shot using all-purpose feature matching. *arXiv preprint arXiv:2305.13310*, 2023.

Zhuoming Liu, Hao Ding, Huaping Zhong, Weijia Li, Jifeng Dai, and Conghui He. Influence selection for active learning. In *Int. Conf. Comput. Vis.*, pages 9274–9283, 2021.

Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Timo Lüddecke and Alexander Ecker. Image segmentation using text and image prompts. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7086–7096, 2022.

Wenjie Luo, Alex Schwing, and Raquel Urtasun. Latent structured active learning. *Adv. Neural Inform. Process. Syst.*, 26, 2013.

Dwarikanath Mahapatra, Behzad Bozorgtabar, Jean-Philippe Thiran, and Mauricio Reyes. Efficient active learning for image classification and segmentation using a sample selection and conditional generative adversarial network. In *Int. Conf. Med. Image Comput. Comput.-Assist. Interv.*, pages 580–588. Springer, 2018.

Hieu T Nguyen and Arnold Smeulders. Active learning using pre-clustering. In *Int. Conf. Learn. Represent.*, page 79, 2004.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. pages 19920–19930, 2020.

Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R Zaiane, and Martin Jagersand. U2-net: Going deeper with nested u-structure for salient object detection. *Pattern recognition*, 106:107404, 2020.

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 *Proc. Int. Conf. Mach. Learn.*, pages 8748–8763. PMLR, 2021.

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10684–10695, 2022a.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10684–10695, 2022b.

Alvin E Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Adv. Neural Inform. Process. Syst.*, 35:36479–36494, 2022.

Akanksha Saran, Safoora Yousefi, Akshay Krishnamurthy, John Langford, and Jordan T. Ash. Streaming active learning with deep neural networks. In *Proc. Int. Conf. Mach. Learn.*, pages 30005–30021. PMLR, 2023.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *Int. Conf. Learn. Represent.*, 2018.

Alex Shonenkov, Misha Konstantinov, Daria Bakshandaeva, Christoph Schuhmann, Ksenia Ivanova, and Nadiia Klokova. Deepfloyd-if, 2023.

Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 761–769, 2016.

Yukun Su, Jingliang Deng, Ruizhou Sun, Guosheng Lin, Hanjing Su, and Qingyao Wu. A unified transformer framework for group-based segmentation: Co-segmentation, co-saliency detection and video salient object detection. *IEEE Transactions on Multimedia*, 2023.

Yoad Tewel, Rinon Gal, Gal Chechik, and Yuval Atzmon. Key-locked rank one editing for text-to-image personalization. *ACM SIGGRAPH 2023 Conference Proceedings*, 2023.

Alexander Vezhnevets, Joachim M Buhmann, and Vittorio Ferrari. Active learning for semantic segmentation with expected change. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3162–3169. IEEE, 2012.

Weiyao Wang, Matt Feiszli, Heng Wang, Jitendra Malik, and Du Tran. Open-world instance segmentation: Exploiting pseudo ground truth from learned pairwise affinity. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4422–4432, 2022.

Weijia Wu, Yuzhong Zhao, Hao Chen, Yuchao Gu, Rui Zhao, Yefei He, Hong Zhou, Mike Zheng Shou, and Chunhua Shen. Datasetdm: Synthesizing data with perception annotations using diffusion models. *Adv. Neural Inform. Process. Syst.*, 2023a.

Weijia Wu, Yuzhong Zhao, Mike Zheng Shou, Hong Zhou, and Chunhua Shen. Diffumask: Synthesizing images with pixel-level annotations for semantic segmentation using diffusion models. In *Int. Conf. Comput. Vis.*, pages 1206–1217, 2023b.

Jiahao Xie, Wei Li, Xiangtai Li, Ziwei Liu, Yew Soon Ong, and Chen Change Loy. Mosaicfusion: Diffusion models as data augmenters for large vocabulary instance segmentation. *arXiv preprint arXiv:2309.13042*, 2023.

Lihe Yang, Xiaogang Xu, Bingyi Kang, Yinghuan Shi, and Hengshuang Zhao. Freemask: Synthetic images with dense annotations make stronger segmentation models. In *NeurIPS*, 2023.

Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. *Adv. Neural Inform. Process. Syst.*, 31, 2018.

Haobo Yuan, Xiangtai Li, Chong Zhou, Yining Li, Kai Chen, and Chen Change Loy. Open-vocabulary sam: Segment and recognize twenty-thousand classes interactively. *arXiv preprint arXiv:2401.02955*, 2024.

Yi Ke Yun and Weisi Lin. Selfreformer: Self-refined network with transformer for salient object detection. *arXiv preprint arXiv:2205.11283*, 2022.

Renrui Zhang, Xiangfei Hu, Bohao Li, Siyuan Huang, Hanqiu Deng, Yu Qiao, Peng Gao, and Hongsheng Li. Prompt, generate, then cache: Cascade of foundation models makes strong few-shot learners. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 15211–15222, 2023.

Yuxuan Zhang, Huan Ling, Jun Gao, Kangxue Yin, Jean-Francois Lafleche, Adela Barriuso, Antonio Torralba, and Sanja Fidler. Datasetgan: Efficient labeled data factory with minimal human effort. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10145–10155, 2021.

Hanqing Zhao, Dianmo Sheng, Jianmin Bao, Dongdong Chen, Dong Chen, Fang Wen, Lu Yuan, Ce Liu, Wenbo Zhou, Qi Chu, Weiming Zhang, and Nenghai Yu. X-paste: Revisiting scalable copy-paste for instance segmentation using clip and stablediffusion. In *Proc. Int. Conf. Mach. Learn.*, 2023.

Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Probabilistic two-stage detection. *arXiv preprint arXiv:2103.07461*, 2021.

Muzhi Zhu, Hengtao Li, Hao Chen, Chengxiang Fan, Weian Mao, Chenchen Jing, Yifan Liu, and Chunhua Shen. Segprompt: Boosting open-world segmentation via category-level prompt learning. In *Int. Conf. Comput. Vis.*, pages 999–1008, 2023.

Xueyan Zou, Jianwei Yang, Hao Zhang, Feng Li, Linjie Li, Jianfeng Wang, Lijuan Wang, Jianfeng Gao, and Yong Jae Lee. Segment everything everywhere all at once. In *Adv. Neural Inform. Process. Syst.*, 2023.

A. Implementation Details

A.1. Dataset

We choose LVIS (Gupta et al., 2019) as the dataset for our experiments. LVIS is a large-scale instance segmentation dataset, comprising approximately 160,000 images with over 2 million high-quality instance segmentation annotations across 1203 real-world categories. The dataset is further divided into three categories: rare, common, and frequent, based on their occurrence across images. Instances marked as 'rare' appear in 1-10 images, 'common' instances appear in 11-100 images, whereas 'frequent' instances appear in more than 100 images. The overall dataset exhibits a long-tail distribution, closely resembling the data distribution in the real world, and is widely applied under multiple settings, including few-shot segmentation (Liu et al., 2023) and open-world segmentation (Zhu et al., 2023; Wang et al., 2022). Therefore, we believe that selecting LVIS allows for a better reflection of the model's performance in real-world scenarios. We use the official LVIS dataset splits, with about 100,000 images in the training set and 20,000 images in the validation set.

A.2. Data Generation

Our data generation and annotation process is consistent with Zhao et al. (2023), and we briefly introduce it here. We first use StableDiffusion V1.5 (Rombach et al., 2022a) (SD) as the generative model. For the 1203 categories in LVIS (Gupta et al., 2019), we generate 1000 images per category, with image resolution 512 $\times$ 512. The prompt template for generation is "a photo of a single $\{CATEGORY\_NAME\}$". We use U2Net (Qin et al., 2020), SelfReformer (Yun and Lin, 2022), UFO (Su et al., 2023), and CLIPseg (Lüddecke and Ecker, 2022) respectively to annotate the raw generative images, and select the mask with the highest CLIP score as the final annotation. To ensure data quality, images with CLIP scores below 0.21 are filtered out as low-quality images. During training, we also employ the instance paste strategy provided by Zhao et al. (2023) for data augmentation. For each instance, we randomly resize it to match the distribution of its category in the training set. The maximum number of pasted instances per image is set to 20.

In addition, to further expand the diversity of generated data and make our research more universal, we also use other generative models, including DeepFloyd-IF (Shonenkov et al., 2023) (IF) and Perfusion (Tewel et al., 2023) (PER), with 500 images per category per model. For IF, we use the pre-trained model provided by the author, and the generated images are the output of Stage II, with a resolution of 256$\times$256. For PER, the base model we use is StableDiffusion V1.5. For each category, we fine-tune the model using the images croped from the training set, with 400 fine-tuning steps. We use the fine-tuned model to generate images.

Table 7. Comparison of different generated data.

Method	Generated Data	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}	AP_c^{box}	AP_c^{mask}	AP_f^{box}	AP_f^{mask}
Baseline	SD	34.00	30.33	24.48	22.65	32.71	29.27	39.62	34.89
Baseline	SD + IF	34.15	30.39	26.12	23.76	32.40	28.97	39.62	34.89
Baseline	SD + IF + Per	34.20	30.39	24.33	22.21	33.23	29.57	39.63	34.89
BSGAL	SD	34.82	31.21	26.76	24.84	33.28	30.01	40.08	35.34
BSGAL	SD + IF	35.13	31.34	26.83	24.32	33.92	**30.57**	**40.13**	35.29
BSGAL	SD + IF + Per	**35.40**	**31.56**	**27.95**	**25.43**	**34.14**	30.56	40.07	**35.37**

We also explore the effect of using different generated data on the model performance (see Table 7). We can see that based on the original StableDiffusion V1.5, using other generative models can bring some performance improvement, but this improvement is not obvious. Specifically, for specific frequency categories, we found that IF has a more significant improvement for rare categories, while PER has a more significant improvement for common categories. This is likely because IF data is more diverse, while PER data is more consistent with the distribution of the training set. Considering that the overall performance has been improved to a certain extent, we finally adopt the generated data of SD + IF + PER for subsequent experiments.

A.3. Model Training

Follow Zhao et al. (2023), We use CenterNet2 (Zhou et al., 2021) as our segmentation model, with ResNet-50 (He et al., 2016) or Swin-L (Liu et al., 2022) as the backbone. For ResNet-50, the maximum training iteration is set to 90,000 and the model is initialized with weights first pretrained on ImageNet-22k then finetuned on LVIS (Gupta et al., 2019), as Zhao

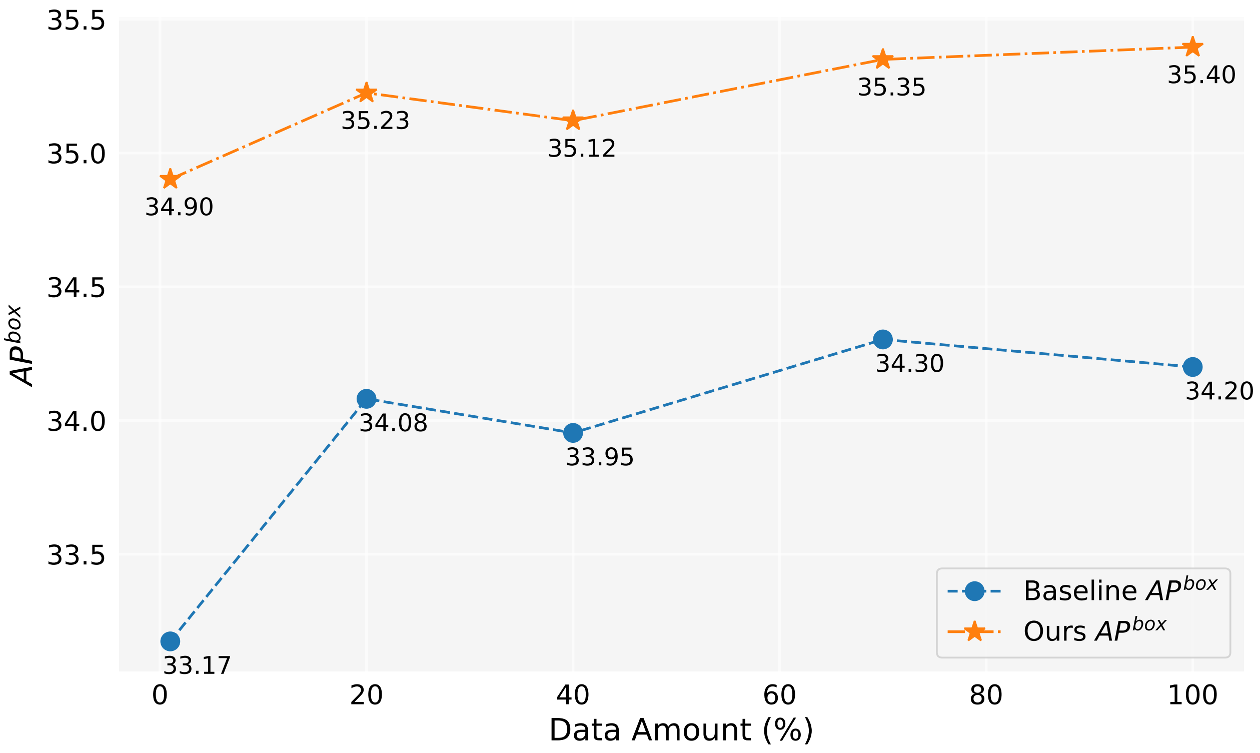

Figure 5. Model performances when using different amount of generated data.

et al. (2023) did. And we use 4 Nvidia 4090 GPUs with a batch size of 16 during training. As for Swin-L, the maximum training iteration is set to 180,000 and the model is initialized with weights pretrained on ImageNet-22k, since our early experiments show that this initialization can bring a slight improvement compared to the weights trained with LVIS. And we use 4 Nvidia A100 GPUs with a batch size of 16 for training. Besides, due to the large number of parameters of Swin-L, the additional memory occupied by saving the gradient is large, so we actually use the algorithm in Algorithm 2.

The other unspecified parameters also follow the same settings as X-Paste (Zhao et al., 2023), such as the AdamW (Loshchilov and Hutter, 2017) optimizer with an initial learning rate of $1e{-4}$.

A.4. Data Amount

In this work, we have generated over 2 million images. Figure 5 shows the model performances when using different amount of generated data(1%,10%,40%,70%,100%). Overall, as the amount of generated data increases, the performance of the model also improves, but there is also some fluctuation. Our method is always better than the baseline, which proves the effectiveness and robustness of our method.

A.5. Contribution Estimation

As mentioned in Section 4.2, we use $\nabla_\theta(L_{\widehat{\mathcal{R}}_b}(\theta) - L_{\mathcal{R}_b}(\theta))^T \nabla_\theta L_{\mathcal{U}_b}(\theta)$ to estimate the contribution of $\mathcal{G}_b$.

Here we actually consider both the direction and the magnitude of the gradient. It is worth mentioning that many previous works actually mainly consider the magnitude of the gradient, for example, the data with large gradient magnitude has more information and should be annotated (Cai et al., 2013) or the wrong outlier data should be filtered (Pruthi et al., 2020).

If we only consider the direction, it is equivalent to normalizing each gradient first, and then calculating, then our calculation formula becomes $\mathcal{G}_b = \frac{\nabla_\theta(L_{\widehat{\mathcal{R}}_b}(\theta) - L_{\mathcal{R}_b}(\theta))^T \nabla_\theta L_{\mathcal{U}_b}(\theta)}{\|\nabla_\theta(L_{\widehat{\mathcal{R}}_b}(\theta) - L_{\mathcal{R}_b}(\theta))\|_2 \|\nabla_\theta L_{\mathcal{U}_b}(\theta)\|_2}$.

Thus, we essentially calculate the cosine similarity. Then we conducted an experimental comparison, as shown in Table 8,

Table 8. Comparison of using grad normalization or not.

Normalize	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}	AP_c^{box}	AP_c^{mask}	AP_f^{box}	AP_f^{mask}
$\times$	35.05	31.27	26.78	24.34	33.79	30.32	40.10	35.39
$\surd$	**35.40**	**31.56**	**27.95**	**25.43**	**34.14**	**30.56**	**40.07**	**35.37**

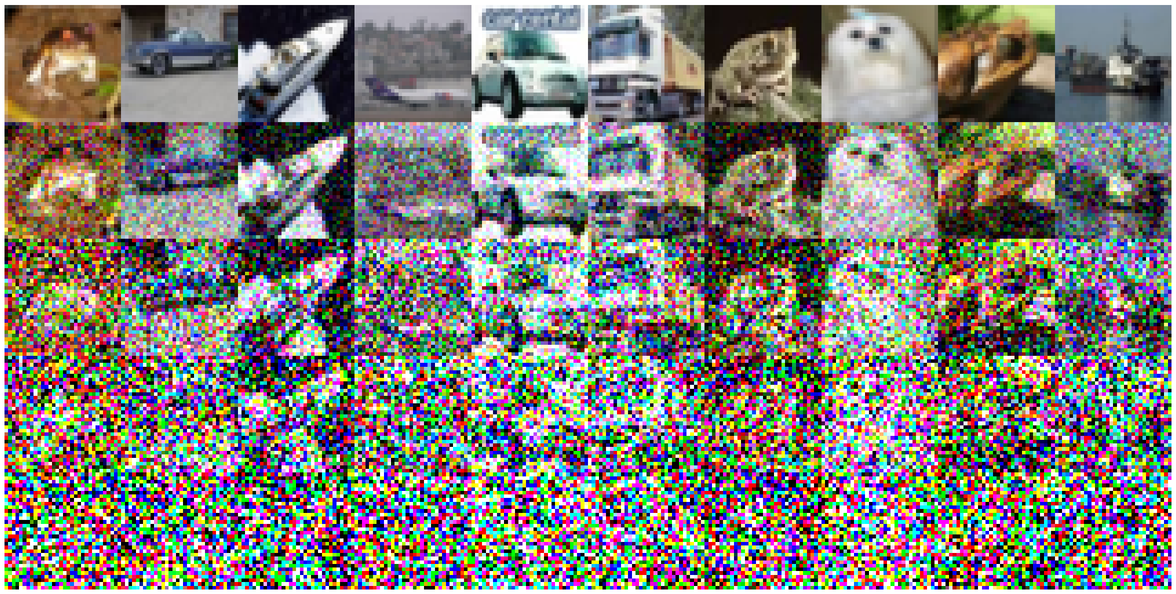

Figure 6. Illustration of noisy images exhibiting various noise scales and categories. Each row, from top to bottom, signifies different noise levels, specifically 0, 40, 100, 200, and 400, respectively. All images are sourced from the CIFAR-10 dataset.

we can see that if we normalize the gradient, our method will have a certain improvement. In addition, since we need to keep two different thresholds, it is difficult to ensure the consistency of the acceptance rate. So we adopt a dynamic threshold strategy, pre-set an acceptance rate, maintain a queue to save the contribution of the previous iter, and then dynamically adjust the threshold according to the queue, so that the acceptance rate stays at the pre-set acceptance rate.

A.6. Toy Experiment

The following are the specific experimental settings implemented on CIFAR-10: We employed a simple ResNet18 as the baseline model and conducted training over 200 epochs, and the accuracy after training on the original training set is 93.02%. The learning rate is set at 0.1, utilizing the SGD optimizer. A momentum of 0.9 is in effect, with a weight decay of 5e-4. We use a cosine annealing learning rate scheduler. The constructed noisy images are depicted in Figure 6. A decline in image quality is observed as the noise level escalates. Notably, when the noise level reaches 200, the images become significantly challenging to identify. For Table 1, we use Split1 as R, while G consists of ‘Split2 + Noise40’, ‘Split3 + Noise100’, ‘Split4 + Noise200’,

A.7. A Simplification Only Forward Once

In Section 4.2, we actually need one more forward on $\mathcal{R}_b$ compared to our baseline. However, we can simplify it to only one forward. The specific reason is that as mentioned in Table 3, we only use the classification loss, L_{cls} this loss is actually the sum of the cross entropy loss of each instance, and whether this instance is generated or real is known during the training process. so the loss can be further distangled as $L_{cls} = L_{real} + L_{gen} + L_{neg}$, where L_{real} is the loss of real instances, L_{gen} is the loss of generated instances, and L_{neg} is the loss of negative instances. Consequently, we can use $\nabla_\theta L_{gen}$ to replace $\nabla_\theta(L_{\widehat{\mathcal{R}}_b}(\theta) - L_{\mathcal{R}_b}(\theta))$

B. More ablations

Momentum Coefficient β. In Algorithm 3, we introduce a momentum coefficient β to update the grad cache. Here we explore the effect of different β on the model performance. A larger beta signifies a greater focus on global information, while a smaller beta indicates a higher attention to the current test batch $\mathcal{U}_b$. Detailed results are presented in Table 9. Observations suggest that when β is 0.1, the performance is the best, which is also the β we finally adopted.

Table 9. Comparison of different β for updating grad cache.

β	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}
0.05	35.14	31.24	27.24	24.50
0.10	**35.40**	**31.56**	**27.95**	**25.43**
0.30	34.84	31.18	26.12	24.32
0.50	34.87	31.03	25.80	22.88
0.80	34.50	30.72	24.37	21.89

Contribution threshold τ. In Algorithm 3, we incorporate a contribution threshold τ, intended for filtering the produced data. Here we investigate the impact of varying values of τ on the model's performance. The larger τ implies a stricter filtration of the generated data, while the smaller τ signifies a looser filtering of the generated data. The specific results are shown in Table 10. We can see the performance is optimal when τ equals -0.05, which is also the τ we eventually settle on for our final model.

Table 10. Comparison of different τ for filtering generated data.

τ	AP^{box}	AP^{mask}	AP_r^{box}	AP_r^{mask}
-0.10	34.68	30.80	26.60	24.40
-0.05	**35.40**	**31.56**	**27.95**	**25.43**
0.00	34.72	30.98	24.96	22.79
0.05	34.29	30.55	23.75	21.71

Online learning vs. Offline learning We compare online learning and offline learning under different iterations. The result is shown in Figure 9.

C. Discussion

C.1. Comparing with existing methods

Table 11. Comparing with existing methods

Method	Data Scale	Downstream	Cost	Quality	Domain Diff
Traditional Active Learning	Limited	$\surd$	High	$\surd$	$\surd$
Generated Data Filtering Methods	Unlimited	$\times$	Low	$\times$	$\times$
Generative Active Learning	Unlimited	$\surd$	Low	$\times$	$\surd$

We've drawn the Table 11, analyzing our setting compared to previous active learning or generative data filtration methods. We've conducted analysis from aspects of data scale, whether it's oriented towards downstream tasks, label quality, labeling costs, and whether there exists domain difference (between generated and real data).

C.2. Analysis of the computational cost

We recorded the training duration and GPU memory usage for training 90,000 iterations with 4 Nvidia 4090 GPUs. It can be observed that our method based on Grad cache increases the GPU memory usage compared to Loss estimate, but it significantly reduces the training time. Compared with our Baseline, the additional time and memory overheads are within an acceptable range.

Table 12. Analysis of the computational cost of different algorithms

Methods	Total Time	Max Memory per GPU
Baseline	17h	6534M
Loss estimate (Algorithm 2)	31h	7114M
Grad cache (Algorithm 3)	21h	9836M

C.3. Future work

We hope that this paper can provide more inspiration to the academic community on how to utilize generated data and how to design better data analysis methods. It should be pointed out that our method is not limited to specific tasks or specific model architectures. In this work, for the convenience of comparison with the baseline, we use the same dataset and model architecture as the baseline. We hope that in future work, we can further verify it on more tasks and model architectures. At the same time, we can also design more flexible and controllable evaluation functions to better utilize generated data. For example, in this paper, when filtering the data with a gradient, there is a trade-off between diversity and consistency. For rare categories in the data, due to the small number of real data itself, diversity should be considered more, while for common categories, due to the large number of real data itself, consistency should be considered more. Therefore, in the future, we can consider adopting a dynamic strategy for different categories. In the long run, our current research is done under the premise of a fixed generative model. A more ideal situation is to involve the generative model in this loop, further optimizing the generative model based on the downstream model's feedback, to achieve a true "generative model in the loop".

D. Visualization

D.1. Selected and Discarded Samples

We show some samples selected and discarded by our method in Figure 7. Our proposed method is able to select high-quality samples (best sample) while filtering out low-quality samples (worst sample), which can effectively improve the data learning efficiency of the model. For example, our method is capable of identifying accurately segmented data for applesauce. In cases where applesauce is not present in the generated raw image or is not encompassed within the segmentation mask, our method can discard such samples. For alarm clocks, our method tends to choose images with more complex appearances.

D.2. Instance Augmentation

We present some augmented data in Figure 8. By randomly pasting generated samples onto the LVIS training set, we effectively enrich the complexity of the scenes and thus increase the model's learning efficiency on the generated data.

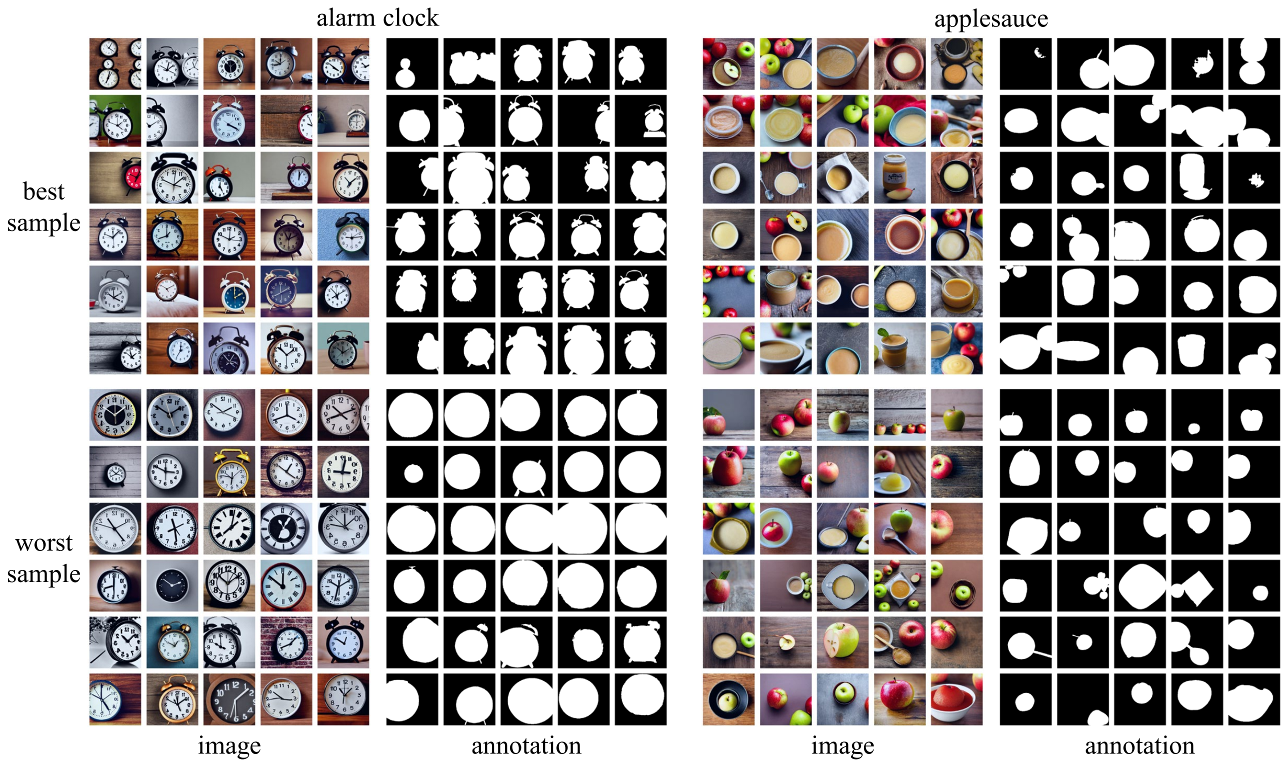

Figure 7. Examples of selected and discarded samples.

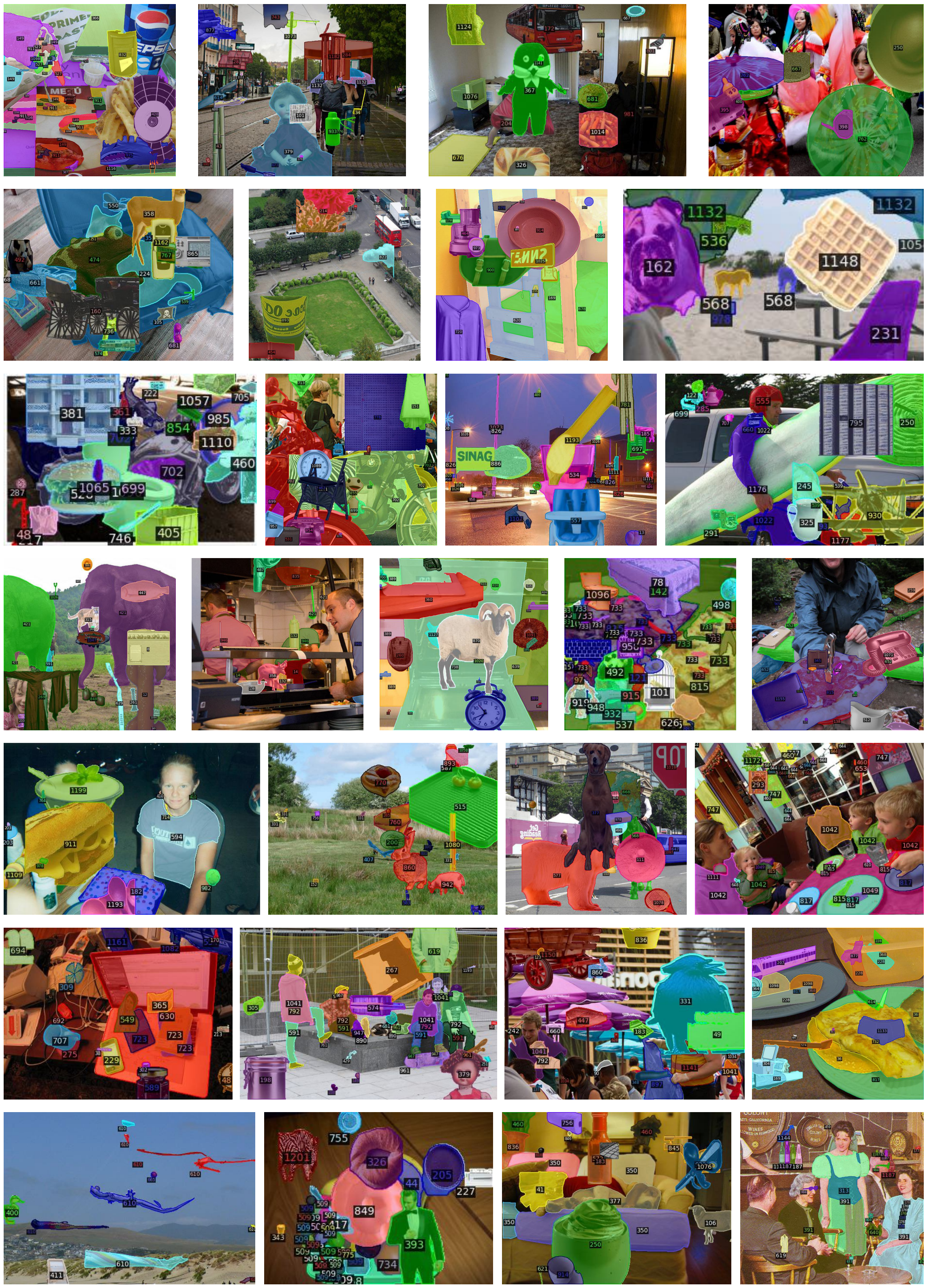
Figure 8. Examples of augmented data.

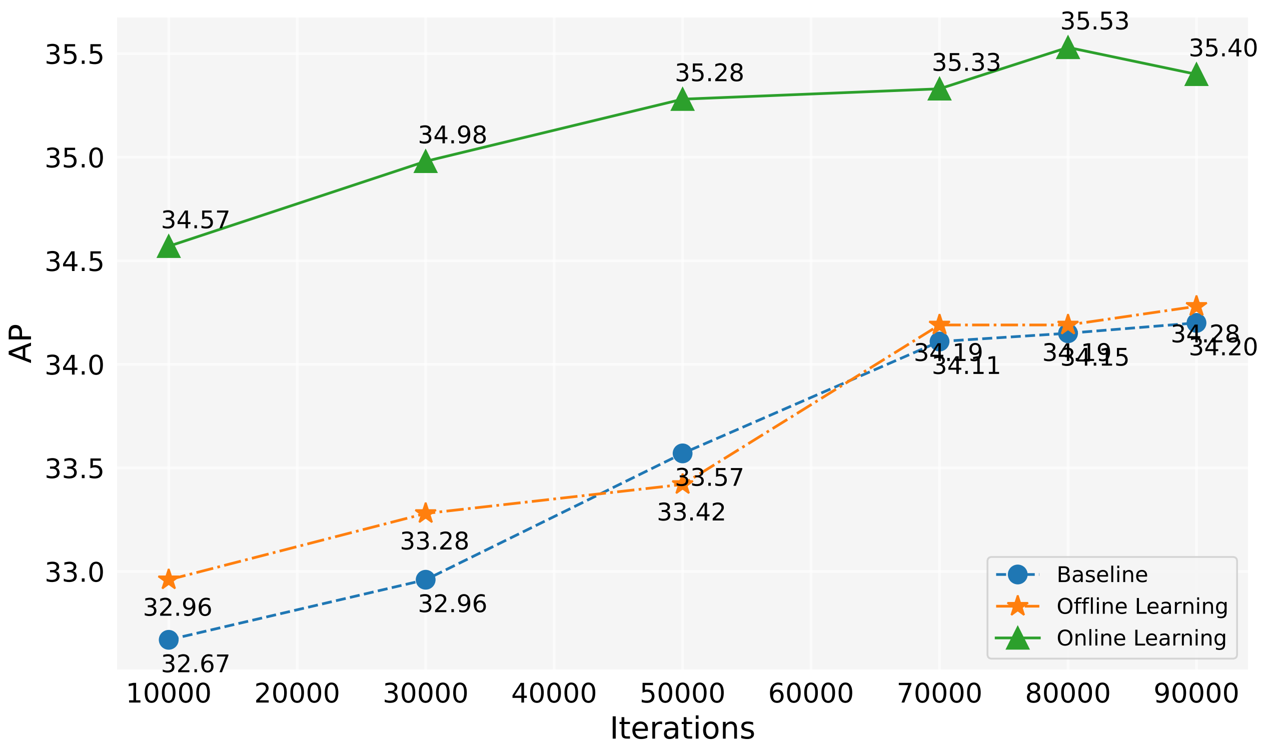

Figure 9. Performance of the model under different iterations.