A Deduplication

We follow SemDeDup (Abbas et al., 2023) in order to deduplicate the dataset. SemDeDup deduplicates LAION by clustering the image embeddings of a pretrained model, and subsequently removing samples within a certain similarity threshold. We choose the threshold value for SemDeDup manually so that we reach the targeted dataset size. We follow the paper and keep 60%-80% of the data (63% for LAION and 80% for DataComp) as this range of values was shown to perform the best on the LAION dataset. For the k-means clustering step of SemDeDup we use 50,000 clusters for the LAION-CAT-440M dataset and 30,000 clusters for the DataComp Medium dataset. We did not tune the number cluster parameters as Abbas et al. (2023) show that it has a small effect on SemDeDup. We refer the reader to Abbas et al. (2023) for more details about the SemDeDup method.

B Details on k-means Clustering

We use the Faiss library (Johnson et al., 2019) for clustering the embeddings on a single GPU. We normalize the embeddings to have a unit length and run spherical k-means using Faiss. In all experiments, we run 100 clustering iterations. We found that 100 iterations are enough as the centroids do not change after this number of iterations.

C Python code for Density-Based Pruning

We include Python-code to solve the quadratic program defined in Eq. 3 in Table 5. The code to calculate $\mathrm{d}_{\mathrm{inter}}$ and $\mathrm{d}_{\mathrm{intra}}$ can be found in Table 6.

D Pretrained Models for calculating embeddings for k-means clustering

Distilled DINOV2-L/14: We use a distilled DINOV2-L/14 model from Oquab et al. (2023). The model is distilled from DINOV2 and has 300M parameters. We resize the images of the LAION or the DataComp datasets to the size of 224x224 and take the output of the last layer of the model. Each image is embedded into a vector of size 1024.

BLIP ViT-B/16: We use the BLIP model to generate a multimodal representation of each image-caption pair in the data. We use the BLIP ViT-B/16 model introduced in Li et al. (2022a). The model has 233M parameters and has been pretrained on a dataset of 129M examples. To embed an image-caption pair, we first embed the image using the Image Encoder of BLIP into a vector of size 768. Then we condition the Image-Grounded Text Encoder of the model on the image embedding and embed the caption. We take the average of the token embeddings (each of size 768) of the last layer of the model as an embedding.

Sentence BERT: Sentence-BERT is a siamese BERT architecture introduced in Devlin et al. (2019). Our motivation behind using this model is the fact that the model learns to maximize the cosine similarity between embeddings of semantically meaningful sentences using a contrastive learning objective. Namely, we use the "all-MiniLM-L6-v2" Sentence BERT model from HuggingFace. This model has been trained on 1B sentence pairs dataset. The model maps each caption onto a 384-dimensional vector. This vector is the output of an average pooling layer applied on top of the last layer of the BERT model.

CLIP ViT-B/16 Encoder We embed the images using OpenAI's CLIP-B/16 model (Radford et al., 2021a) by mapping each image into a 512-dimensional vector using the Vision Transformer Encoder of the CLIP model. This vector is the representation of the CLS token in the output layer of the model.

CLIP B/16 Text Encoder We embed the captions using OpenAI's CLIP-B/16 (Radford et al., 2021a) model by mapping each caption into a 512-dimensional vector using the Text Encoder of the CLIP model. This vector is the representation of the last token in the output layer of the model.

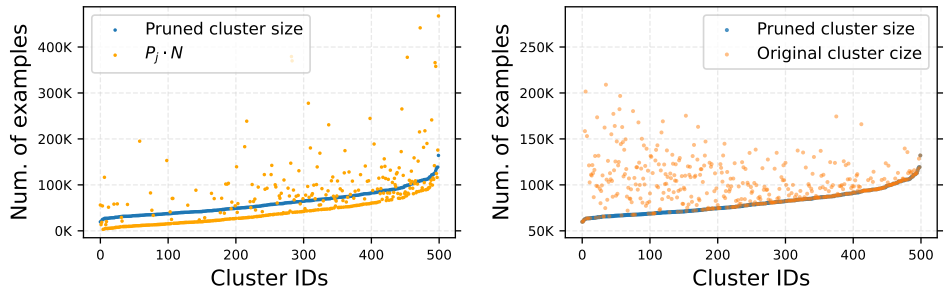

Figure 7: (left) Pruned cluster size vs $\mathrm{P}_j N$ after solving the quadratic program. (right) Pruned cluster size vs original cluster size. We that the method tends to remove more examples from large clusters resulting in a more cluster-balanced dataset. In both plots, the clusters are sorted in the x-axis by the pruned cluster size. The results are for filtering the LAION-50M dataset down to 30M examples using the DINOV2-L/14 embeddings.

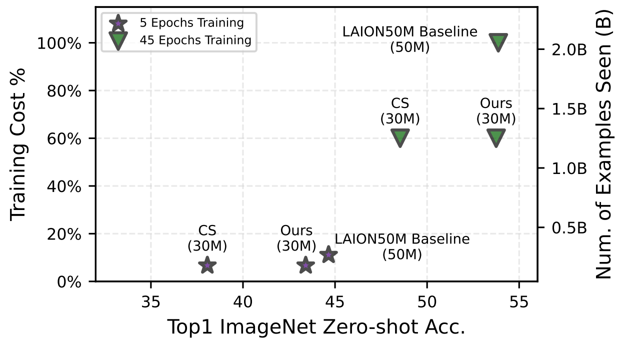

Figure 8: Performance grows consistently with continued training and we close the gap to training on the full LAION-50M dataset when training for 45 epochs, despite only using 30M samples. We also outperform the LAION CLIP-B/16 score (CS) filtering.

E Training hyperparameters

We include the training hyperparameters in Table 7.

F Additional results

We show results for continued training in Fig.8. For this, we train the same models for five and forty-five epochs on the LAION-50M subset, and on 30M examples filtered from it using our pipeline. We consistently outperform CLIP score filtering (CS) throughout training and even close the gap to training on the full LAION-50M dataset when training for forty-five epochs.

We show consistent improvements of DBP over SSP-Pruning in Table 8.

G Software stack

We use different open-source software packages for our experiments, most notably SLURM (Yoo et al., 2003), OpenCLIP (Ilharco et al., 2021), scipy and numpy (Virtanen et al., 2020), GNU parallel (Tange, 2011), Faiss (Johnson et al., 2019), PyTorch (Paszke et al., 2017) and torchvision (Marcel & Rodriguez, 2010).

Table 5: Python code for the quadratic program solver

```python

import numpy as np
import torch
from qpsolvers import solve_qp

# Input: d_inter (List), d_intra (List), temp (float), num_centroids (int
    ), filtered_dataset_size (int), num_items_in_each_cluster (List)

# Output: X (list) <- Number of samples per cluster

softmax = torch.nn.Softmax()
probs = softmax( (d_inter *  d_intra)/temp )
P = np.eye(num_centroids)
q = - probs * filtered_dataset_size
A = np.array(1.0 * num_centroids)
b = np.array([filtered_dataset_size])

# Define the lower and upper bounds
min_samples = 1
bounds = np.array([ ( min_samples, num_items_in_each_cluster[i] )
                     for i in range(num_centroids) ]

X = solve_qp(P=P, q=q, A=A, b=b,
             lb=bounds[:,0], ub=[:,1], solver='osqp')

X = np.rint(X).astype(int)
```

Table 6: Python code for computing $\mathrm{d_{inter}}$ and $\mathrm{d_{intra}}$.

```python

import numpy as np
import faiss

# Input: norm_embs (array), emb_dim (int), num_centroids (int),
    filtered_dataset_size (int), niter (int), seed (int), num_NNs (int)

# Output: d_intra (list), d_inter (list)

# Cluster the data
kmeans = faiss.kmeans(dim, num_centroids, niter=niter, seed=seed,
                      spherical=True, gpu=True, verbose=True)
kmeans.train(norm_embs)

# Compute d_intra
sim_to_centroid, nearest_cent = kmeans.index.search(norm_embs, 1)

d_intra = []
for cluster_id in range(num_centroids):
    cluster_item_ids = np.where( nearest_cent==cluster_id )
    cluster_d_intra = ( 1 - sim_to_centroid[cluster_item_ids] ).mean()
    d_intra.append(cluster_d_intra)

# Compute d_inter
sim_to_NN_centroids = kmeans.index.search( kmeans.centroids, num_NNs+1 )
dist = 1 - sim_to_NN_centroids[:, 1:]
d_inter = np.mean( dist, axis=1 )
```

Table 7: Training parameters for CLIP. We follow the standard hyperparameters used for each dataset. We use the OpenCLIP hyperparameters for experiments on the LAION dataset and the DataComp hyperparameters for experiments on the DataComp Medium dataset.

Parameter	Value
Model	CLIP ViT-B-32
Warmup (LAION)	2000 training steps
Warmup (DataComp)	500 training steps
Batch size (LAION)	33,792
Batch size (DataComp)	4,096
Learning rate	5.0e-4, cosine scheduler
Optimizer	AdamW, wd=0.2, betas=(0.9, 0.98), eps=1.0e-6

Table 8: Density-based pruning (DBP) helps improve the performance of SSP-Pruning. We duplicate the LAION-CAT-440M dataset to 277M examples and then apply SSP pruning or DBP to filter the dataset to 112M or 166M examples and train CLIP-B/32 on them for 32 epochs. We report the average zero-shot accuracy on 38 datasets from Gadre et al. (2023). We set the cluster balancing value of SSP-Pruning method to 1.0.

Method/ Dataset size	112M (3.6B examples seen)	166M (5.3B examples seen)
DBP	49.8	51.6
SSP-Pruning	48.6	50.7