A Implementing data-parallel distribution

CLIP training benefits from a large batch size [35, 7]. FLYT trains a CLIP model while weighting each example compared to others in the batch, which also requires a sufficiently large batch size. However, training FLYT requires more memory than training a standard CLIP model since we have to keep the computational graph from ϕ to $\hat{L}_{\text{down}}$ in order to calculate $\nabla_\phi \hat{L}_{\text{down}}$. This means that in order to train FLYT on a large scale, we need to implement data-parallel distributed training.

Our goal is to compute $\nabla_\phi \hat{L}_{\text{down}}$ using data parallelism across multiple GPUs, which is not trivial considering FLYT introduces different dependencies between examples in the batch. We start by using the chain rule to write:

$$\nabla_\phi \hat{L}_{\text{down}} = \left[\frac{\partial w_{1:B}}{\partial \phi}\right] \nabla_{w_{1:B}} \hat{L}_{\text{down}},$$

and focus the second factor

$$\nabla_{w_{1:B}} \hat{L}_{\text{down}} = \nabla_{w_{1:B}} \ell_{\text{DS}}(f_{\theta_+(w_{1:B})}(\hat{z}_{1:B'})),$$

where

$$\theta_+(w_{1:B}) = \text{update}(g(w_{1:B}))$$

with

$$g(w_{1:B}) = \nabla_\theta \ell_{\text{WCL}}(f_\theta(z_{1:B}), w_{1:B})$$

and $\text{update}(\cdot)$ is an optimizer update function. We then use the chain rule again to write:

$$\nabla_{w_{1:B}} \hat{L}_{\text{down}} = \left[\frac{\partial g}{\partial w_{1:B}}\right] \nabla_g \ell_{\text{DS}}(f_{\text{update}(g)}(\hat{z}_{1:B'})),$$

We may further apply the chain rule (and the fact the partial derivatives commute) to write

$$g = \sum_{i\in[B]} \frac{\partial \ell_{\text{WCL}}(f_{1:B}, w_{1:B})}{\partial f_i} \nabla_\theta f_\theta(z_i)$$

where f_i is shorthand for the embedding $f_\theta(z_i)$ that implicitly says we only need to store the value of the embedding and not the computational graph behind it. This way each GPU calculates g for its own sub-batch, and we can then gather and sum their result to get g for the full batch using the CLIP gradient accumulation technique from Cherti, Mehdi et al. [7]. Using this, we define

$$v \triangleq \nabla_g \ell_{\text{DS}}(f_{\text{update}(g)}(\hat{z}_{1:B'})),$$

which we can again compute using gradient accumulation. Treating v as a constant vector independent of $w_{1:B}$, the gradient we would like to calculate is

$$\nabla_{w_{1:B}} \hat{L}_{\text{down}} = \frac{\partial \langle g, v\rangle}{\partial w_{1:B}} = \frac{\partial}{\partial w_{1:B}} \sum_{i\in[B]} \frac{\partial \ell_{\text{WCL}}(f_{1:B}, w_{1:B})}{\partial f_i} \langle \nabla_\theta f_\theta(z_i), v\rangle.$$

If we define the function

$$h_i(\eta) = \frac{\partial \ell_{\text{WCL}}(f_{1:B}, w_{1:B})}{\partial f_i} f_{\theta+\eta v}(z_i)$$

Then our gradient is simply

$$\nabla_{w_{1:B}} \hat{L}_{\text{down}} = \sum_{i\in[B]} \frac{\partial}{\partial w_{1:B}} h_i'(0).$$

Which is amenable to gradient accumulation.

The first factor

$$\frac{\partial}{\partial \phi} w_{1:B} = \frac{\partial}{\partial \phi} \text{softmax}(q_\phi(\zeta_{1:B}))$$

can be calculated separately using gradient accumulation, and we get

$$\nabla_\phi \hat{L}_{\text{down}} = \left[\frac{\partial w_{1:B}}{\partial \phi}\right] \nabla_{w_{1:B}} \hat{L}_{\text{down}}.$$

Table 4: DFN-Base reproduction experiments.

Initialization	Num. samples	Notes	Training dataset	ImageNet	Average
Random	10M		CC12M	0.230	0.300
Random	50M		CC12M	0.274	0.324
Random	128M		CC12M	0.281	0.328
OpenAI	128M	DFN-Base	CC12M	**0.297**	0.333
OpenAI	128M		CC12M+CC3M+SS15M	0.286	<u>0.334</u>
OpenAI	512M	2x Batch Size (8192)	CC12M	<u>0.296</u>	**0.338**

Table 5: DFN-FT reproduction experiments

Initialization	Num. samples	Notes	Training dataset	ImageNet	Average
DFN-Base	400K	DFN-FT	IN1K+MS COCO+Flickr30k	**0.342**	**0.348**
DFN-Base	1.6M		IN1K+MS COCO+Flickr30k	<u>0.340</u>	**0.348**
DFN-Base	3M		IN1K+MS COCO+Flickr30k	0.335	0.338
DFN-Base	400K	1/10 LR (5e-5)	IN1K+MS COCO+Flickr30k	0.321	<u>0.344</u>
DFN-Base	1M	1/10 LR (5e-5)	IN1K+MS COCO+Flickr30k	0.324	0.342
DFN-Base	12M	1/10 LR (5e-5)	IN1K+MS COCO+Flickr30k	0.319	0.340
DFN-Base	800K		IN1K	0.333	0.343

B M-FLYT input details

In order to experiment with M-FLYT, we reproduce some existing scoring methods and create some new ones:

1. **CLIP scores.** We use the CLIP scores calculated using OpenAI ViT-B/32 and ViT-L/14 CLIP models [35].
2. **DFN.** The results of Fang et al. [16] are proprietary and we do not have access to their best DFN model or the scores created by it. The authors provided a version that uses only publicly available resources, but it was not trained using all the improved techniques they discovered. Thus, we reproduced their work using only publicly available resources. We use the CLIP score of 2 DFN networks: one before fine-tuning on ImageNet, and one after. Appendix B.1 provides the details of our reproduction.
3. **HYPE.** Kim et al. [22] provided a PyTorch implementation which includes their hyperbolic CLIP weights, and the functionality required to reproduce the image specificity ϵ_i, text specificity ϵ_t, and negative Lorentzian distance $-d_L$. We used these three values as inputs.
4. **s-CLIPLoss.** Wang et al. [49] also provided a PyTorch implementation. We used OpenAI ViT-B/32 embeddings to calculate s-CLIPLoss and $\mathrm{NormSim}_{\infty}(\mathrm{IN1K})$.
5. **Binary classifiers.** We trained two linear binary models that classify between a "high quality" dataset and the DataComp medium scale dataset. We describe them in Appendix B.2 below.
6. **E-FLYT.** We use our E-FLYT-Base scoring model, as detailed in Appendix C.7.

B.1 DFN reproduction

Since M-FLYT requires good inputs to perform well, we made effort to train a high performing DFN. In the paper, Fang et al. [16] trained a ViT-B/32 CLIP model initialized with OpenAI's checkpoint on the high-quality private dataset HQITP-350M. Since we do not have access to this dataset, we trained our DFN on CC12M [41]. Similar to Fang et al. [16], we then fine-tune on the combined training sets of IN1K with sampled OpenAI templates as texts, MS COCO, and Flickr30k. Unless otherwise specified, we use the same hyper-parameters as DataComp medium scale training. Tables 4 and 5 show results we obtained with different hyperparameter choices when our reproductions DFN-Base and DFN-FT, respectively.

Table 6: Detailed hyper parameters of the different configurations used for training and ablations of M-FLYT and E-FLYT. $*$ Initialization "DataComp medium" means we train a model from scratch on the unfiltered DataComp medium scale dataset using the DataComp medium scale training configuration.

Parameter	E-FLYT-Base	E-FLYT-22	E-FLYT-LR	M-FLYT-Base
Optimizer	AdamW	AdamW	AdamW	AdamW
Weight decay	0.2	0.2	0.2	0.2
(β_1, β_2)	$(0.9, 0.98)$	$(0.9, 0.98)$	$(0.9, 0.98)$	$(0.9, 0.98)$
Scheduler	cosine	cosine	cosine	cosine
Reference model LR	5e-5	5e-5	Variable	5e-5
Scoring model LR	1e-3	1e-3	Variable	1e-3
Feature extractor init	OpenAI	OpenAI	OpenAI	None
Reference model init	OpenAI	OpenAI	OpenAI	DataComp medium*
Downstream loss	CE + temperature	CLIP	CE	CE
Downstream batch size B'	3072	768	1536	3072
Upstream batch size B	4096	4096	4096	4096
Num. downstream samples	15.4M	3.8M	7.7M	15.4M
Num. upstream samples	20.5M	20.5M	20.5M	20.5M
Num. steps	5000	5000	5000	5000
Num. warmup steps	100	100	100	100
Downstream dataset	ImageNet	22 tasks	ImageNet	ImageNet
Upstream dataset	DataComp medium	DataComp medium	DataComp medium	DataComp medium
Trainable reference parameters	all	all	all	all
Trainable scoring parameters	all	all	all	all

B.2 Binary classifiers input

When training our top performing M-FLYT, we use two binary linear classifiers. These models take as input embeddings generated by OpenAI's ViT-B/32 model and predict which of two datasets these embeddings come from. IN1K-Classifier classifies between the ImageNet training set and the DataComp medium scale dataset image embeddings. CC2M-Classifier classifies between concatenated image and text embeddings of the DataComp medium scale dataset and *CC2M*. CC2M is a dataset of 2M examples we created by running our IN1K-Classifier on the CC12M dataset [41] and taking the top 2M examples that the model classified as ImageNet examples. As shown in Table 2, while IN1K-Classifier underperforms compared to CLIP score filtering, it still demonstrates improvement over no filtering and receives substantial weighting from M-FLYT. In contrast, CC2M-Classifier achieves performance similar to CLIP score filtering but receives a lower M-FLYT weighting. We trained them for 40K steps with a batch size of 256, a cosine learning rate with a maximal value of 0.01 and 500 warm-up steps.

C Experimental setup

C.1 Experiment configurations

When training M-FLYT and E-FLYT, we optimize the mixing model, the embeddings scoring model, and the reference models using AdamW [26] with a weight decay of 0.2, $(\beta_1, \beta_2) = (0.9, 0.98)$, and a cosine scheduler [25]. Due to computational constraints, some experiments compare against different model configurations. We list their full hyper parameters in Table 6 and describe their main differences here. We refer to the configuration from Section 4.2 as *M-FLYT-Base*, and to the E-FLYT configuration used in the experiments described in Appendix C.7 as *E-FLYT-Base*. *E-FLYT-22* maintains the same parameters as E-FLYT-Base, with three key differences: it employs the standard CLIP loss as the downstream loss, uses a downstream batch size B' of 768, and trains with 22 downstream tasks. *E-FLYT-LR* uses the same configuration as E-FLYT-Base, except it employs the CE loss without temperature and uses a downstream batch size B' of 1536; this configuration was used exclusively for learning rate experiments.

C.2 Computational cost

In terms of FLOPs, M-FLYT requires slightly more than $2\times$ the FLOPs of regular CLIP model training per example, while E-FLYT requires slightly more than $3\times$. Since we only train our models

Table 7: Experiments using the M-FLYT-Base configuration (Table 6).

Experiment	Baseline	Variation	ImageNet	Average
M-FLYT	—	—	0.359	0.371
Downstream loss	CE	CE + temperature	0.352	0.366
Reference model initialization	Trained on DC medium	Random initialization	0.341	0.354
		OpenAI	0.342	0.355
Input standardization	Yes	No	0.323	0.337

Table 8: Comparison of training M-FLYT with all input methods vs leaving one method out. Numbers in parentheses show performance degradation compared to using all inputs. We highlight values with the largest degradation, indicating which methods provide the most benefit when included.

Inputs	ImageNet	ImageNet dist. shifts	VTAB	Retrieval	Average
All Inputs	0.359	0.294	0.383	0.310	0.371
No ViT-B/32, ViT-L/14	0.355 (0.004)	0.288 (0.006)	0.367 (0.016)	0.307 (0.003)	0.360 (0.011)
No DFN-Base, DFN-FT	0.341 (**0.018**)	0.280 (**0.014**)	0.375 (0.008)	0.281 (**0.029**)	0.358 (0.013)
No $-d_L, \epsilon_i, \epsilon_t$	0.351 (0.008)	0.291 (0.003)	0.366 (**0.017**)	0.308 (0.002)	0.364 (0.007)
No s-CLIPLoss, $\mathrm{NormSim}_\infty(\mathrm{IN1K})$	0.354 (0.005)	0.292 (0.002)	0.377 (0.006)	0.307 (0.003)	0.364 (0.007)
No IN1K-Classifier, CC2M-Classifier	0.352 (0.007)	0.288 (0.006)	0.368 (0.015)	0.301 (0.009)	0.355 (**0.016**)
No E-FLYT	0.352 (0.007)	0.292 (0.002)	0.376 (0.007)	0.309 (0.001)	0.366 (0.005)

for 20M examples (compared to DataComp's 128M), M-FLYT uses approximately $3\times$ fewer FLOPs than training a DataComp medium scale model, and E-FLYT uses approximately $2\times$ fewer.

With our implementation, M-FLYT takes 40 A100 hours to train, and E-FLYT takes 45 A100 hours, roughly equivalent to running a DataComp medium scale experiment. For SCS with a batch size $G = 100K$, it takes about 100 minutes to sample 128M examples using a naive Python implementation on a single CPU. For larger scales (DataComp large or Xlarge), the filtering computation becomes negligible relative to the overall training cost.

Comparing to other filtering methods:

- Fang et al. [16] reported (in Table 2) training their ViT-B/32 DFN for 5.12B samples seen, which requires approximately 80-120x more FLOPs than our method
- Kim et al. [22] trained a ViT-L/14 CLIP model on 8 V100 GPUs for 256M samples, taking 488 GPU hours (61 hours $\times$ 8 GPUs)
- Wang et al. [49], which doesn't require additional model training beyond the pretrained OpenAI CLIP model, reported 5 hours of preprocessing on an L40 machine

C.3 M-FLYT ablations

Table 7 shows experiments using M-FLYT. The standard CE loss slightly outperforms CE with temperature (Appendix C.8), and reference model initialized with "DataComp medium" (explained in Appendix C.7) outperforms OpenAI initialization. We also find that input standardization greatly increases performance.

Table 8 shows the result of training M-FLYT while excluding groups of related inputs. Removing any input group degrades performance, with several notable results: DFN reproduction scores have the strongest impact on ImageNet, ImageNet distribution shifts, and retrieval scores (degrading performance by 1.8%, 1.4%, and 2.9% respectively). For VTAB tasks, excluding HYPE scores causes the largest drop (1.7%), while unexpectedly, removing the linear classifiers—despite being the weakest standalone filtering methods—leads to the largest decline in average performance (1.6%).

Table 9: Comparing M-FLYT to the original mixing method of HYPE [22] and s-CLIPLoss [49]. HYPE combines scores by summing their three components ($-d_L + \epsilon_i + \epsilon_t$), while s-CLIPLoss applies a two-stage filtering process: selecting the top 30% based on s-CLIPLoss scores, then selecting the top 20% according to their NormSim scores from this filtered subset.

Input method	Mixing method	ImageNet	Average
HYPE	Original (sum scores)	0.309	0.341
	M-FLYT	0.322	0.347
s-CLIPLoss and NormSim	Original (two stage)	0.331	0.363
	M-FLYT	0.334	0.357

C.4 Comparing M-FLYT to aggregation methods in prior work

Table 9 compares M-FLYT against the original mixing strategies of HYPE and s-CLIPLoss. For HYPE's components, M-FLYT outperforms the simple summation approach on both ImageNet (+1.3%) and average (+0.6%) metrics. When mixing s-CLIPLoss scores, M-FLYT performs similarly to the original two-stage filtering method, showing marginal improvement on ImageNet (0.3%) but slightly lower average performance (0.6%).

C.5 M-FLYT baseline - ImageNet-weighted standardized sum

Here we explain the ImageNet-weighted standardized sum baseline we compare against in Section 4.2. For K ImageNet accuracies $\mathrm{IN}_{1:K}$ corresponding to K scoring methods, we first standardize the per-example scores produced by these methods individually across all training examples to have zero mean and unit standard deviation. We denote by s_i^j the standardized score that method $i \in 1, ..., K$ gave example z_j.

We then normalized the ImageNet accuracies $\mathrm{IN}_{1:K}$ between zero and one, and add a scalar to achieve our desired ratio r:

$$\hat{w}_i = \frac{\mathrm{IN}_i - \min(\mathrm{IN}_{1:K})}{\max(\mathrm{IN}_{1:K}) - \min(\mathrm{IN}_{1:K})},$$
$$w_i = \hat{w}_i + \frac{1}{r-1},$$

where $\min(\mathrm{IN}_{1:K})$ is the lowest ImageNet score, and $\max(\mathrm{IN}_{1:K})$ is the highest. This ensures that the ratio between the maximum and minimum weights equals r:

$$r = \frac{\max(w_{1:K})}{\min(w_{1:K})}.$$

The final score for example z_j is then $\sum_{i=1}^{K} w_i s_i^j$.

In Table 2 we experiment with max-to-min ratios $r \in \{2, 4, 8, 16\}$.

C.6 E-FLYT learning rate tuning

Table 10 compares different scoring model and reference model learning rates using the E-FLYT-LR configuration. We notice that E-FLYT is not very sensitive to the scoring model LR (with reference LR fixed at 1e-4) and select 1e-3 based on slightly better results. E-FLYT is somewhat sensitive to reference model LR, and with scoring model LR fixed at 1e-3, 5e-5 gives the best results. These values are used for subsequent experiments.

C.7 E-FLYT results

For E-FLYT, we use a CLIP ViT-B/32 [35] as the feature extractor and an $\mathrm{FFN}_{\mathrm{GLU}}$ [42] scoring model. We initialize both the feature extractor and the reference model with OpenAI's checkpoint.

In Table 11 we show experiments using E-FLYT-Base. We evaluate three alternatives for the downstream loss function: The CLIP loss, as defined in Equation (1) $\ell_{\mathrm{C}}(f_{\theta_+(w_{1:B})}(\hat{z}_{1:B'}))$, standard cross entropy (CE) loss, and CE with a temperature parameter (see definitions in Appendix C.8). Our experiments show that CLIP loss and CE with temperature demonstrate similar performance, both outperforming the standard CE loss. For subsequent experiments, we use CE with temperature

Table 10: Learning rate tuning using the E-FLYT-LR configuration (Table 6). We first optimize the scoring model LR with reference model LR fixed at 1e-4, then optimize reference model LR with scoring LR fixed at 1e-3.

Scoring model LR	Reference model LR	ImageNet	Average
5e-4	1e-4	0.286	0.300
1e-3	1e-4	0.286	0.306
5e-3	1e-4	0.280	0.310
1e-2	1e-4	0.277	0.300
1e-3	1e-5	0.158	0.238
1e-3	5e-5	**0.292**	**0.319**
1e-3	1e-4	<u>0.286</u>	<u>0.306</u>
1e-3	5e-4	0.267	0.296

because it better resembles the accuracy metric used to evaluate most DataComp tasks. We found it essential to use separate learned temperature parameter for the upstream and downstream loss functions when using either the CLIP or CE with temperature loss. We initialize the downstream temperature parameter τ_{DS} to 1/0.07, as is the default when training a CLIP model from scratch, while the initial value for the upstream temperature is set to 100 as part of the pretrained reference model weights. The downstream temperature is updated during training using $\nabla_{\tau_{\text{DS}}} \hat{L}_{\text{down}}$.

Table 11 also compares reference model and feature extractor initializations, where initializing with stronger models outperform weaker ones. We evaluate four initializations in ascending order of performance: Random initialization, "DataComp medium", where we train a model from scratch on the unfiltered DataComp medium scale dataset using the DataComp medium scale training configuration, OpenAI checkpoint, and "DataComp-13B", which is a model trained by Gadre et al. [17] on the DC-1B dataset for 13B samples seen. Further results in the table show that training the reference model is much better than freezing it, and that fine-tuning the feature extractor prevents the training from succeeding. Our ablation studies show that performance remains stable with downstream batch sizes ranging from 1024 to 4096. Similarly, performance is consistent when training for 1250 to 12500 steps, suggesting that E-FLYT could be optimized using less compute.

As discussed in Section 5, we believe that our E-FLYT training stack has significant room for improvement.

C.8 Downstream loss

We now explain how construct the downstream loss from a downstream batch $\hat{z}_{1:B'}$ containing labeled data of the form $\hat{z}_i = (x_i, c_i)$, where x_i is an image and $c_i \in \{1, \dots, K\}$ is a class label. First, we uniformly sample templates for each of the classes following DataComp's class-specific template definitions $t_{1:K} = \text{sample}(1:K)$. Then, the CE loss is given by

$$\ell_{\text{DS}}(\hat{z}_{1:B'}) = \sum_{i=1}^{B'} -\log\left(\frac{e^{\tau_{\text{DS}}\langle f_\theta(x_i), f_\theta(t_{c_i})\rangle}}{\sum_{j=1}^{K} e^{\tau_{\text{DS}}\langle f_\theta(x_i), f_\theta(t_j)\rangle}}\right),$$

where τ_{DS} is a learnable temperature parameter we only use in the CE+temperature variation for the downstream data. Using the standard CE loss, τ_{DS} is set to 1. Using this notation, we define the downstream CLIP loss mentioned in Appendix C.7 with a separate temperature parameter τ_{DS} as

$$\ell_{\text{C}}(\hat{z}_{1:B'}) = \sum_{i=1}^{B'} -\log\left(\frac{e^{\tau_{\text{DS}}\langle f_\theta(x_i), f_\theta(t_{c_i})\rangle}}{\sum_{j=1}^{B'} e^{\tau_{\text{DS}}\langle f_\theta(x_i), f_\theta(t_{c_j})\rangle}}\right) + \sum_{i=1}^{B'} -\log\left(\frac{e^{\tau_{\text{DS}}\langle f_\theta(t_{c_i}), f_\theta(x_i)\rangle}}{\sum_{j=1}^{B'} e^{\tau_{\text{DS}}\langle f_\theta(t_{c_i}), f_\theta(x_j)\rangle}}\right).$$

C.9 E-FLYT with diverse downstream data

Instead of using the ImageNet training set alone for the downstream data, we experiment with using 22 training sets from the DataComp benchmark. We now detail the training configuration, and then discuss the results of experiments using E-FLYT-22. First, we list the datasets used:

Table 11: Experiments using the E-FLYT-Base configuration (Table 6). $*$ Initialization "Trained on DC-1B" means using the ViT-B-32 model trained by Gadre et al. [17] on DC-1B for 13B samples seen (Hugging Face model card).

Experiment	Baseline	Variation	ImageNet	Average
E-FLYT-Base	—	—	0.316	0.323
Downstream loss	CE + temperature	CE (no temperature)	0.287	0.305
		CLIP	0.313	0.330
Reference model initialization	OpenAI	Random initialization	0.270	0.312
		Trained on DC medium	0.290	0.308
		Trained on DC-1B*	0.314	0.320
Feature extractor initialization	OpenAI	Random initialization	0.157	0.234
		Trained on DC medium	0.287	0.302
		Trained on DC-1B*	0.325	0.329
Reference model + feature extractor initialization	OpenAI	Trained on DC-1B*	0.326	0.331
Trainable parameters	Scoring + reference model	Only scoring model	0.251	0.298
		+ Feature extractor	0.135	0.227
Num. training steps	5000	12500	0.306	0.320
		1250	0.311	0.329
Downstream Batch Size	3072	1024	0.315	0.313
		4096	0.311	0.316

- ImageNet [12]
- MNIST [11]
- SVHN [31]
- CIFAR10 [24]
- CIFAR100 [24]
- Food-101 [4]
- Oxford Flowers-102 [32]
- Oxford-IIIT Pet [33]
- iWildCam [3]
- SUN-397 [50]
- EuroSAT [20]
- Stanford Cars [23]
- DTD [9]
- RESISC45 [6]
- GTSRB [46]
- FGVC Aircraft [30]
- Pascal VOC 2007 [15]
- Country211 [35, 48]
- Rendered SST2 [54]
- FMoW [8]

Table 12: Different downstream tasks experiments using the E-FLYT-22 configuration (Table 6).

Downstream tasks variation	ImageNet	Average
Imbalanced 22 tasks	0.315	0.330
Balanced 22 tasks	0.274	0.311
Only ImageNet	0.313	0.330

Table 13: SCS parameter ablations: Effects of repetition penalties and batch sizes.

Repetition penalty α	Batch size G	ImageNet	Average
0.10	100K	0.395	0.369
0.15	100K	**0.401**	0.377
0.20	100K	0.399	0.377
0.25	100K	0.395	**0.378**
0.30	100K	0.396	0.374
0.40	100K	0.387	0.373
0.50	100K	0.388	0.376
0.60	100K	0.377	0.369
0.15	10K	0.399	0.373
0.15	1M	0.398	0.371

- Dollar Street [38]
- STL-10 [10]

When preparing our datasets, we split them into tar files of 100 examples each in webdataset format. During training, we use a shuffle buffer of 50,000 examples. Since the webdataset loader reads entire tar files at once, small shards and a large shuffle buffer help ensure diverse tasks within each training batch. The combined dataset contains 2M examples, with ImageNet making up the majority. In Table 12, we compare different dataset configurations. Using the combined dataset without balancing classes yields performance very similar to using only the ImageNet dataset. When we balance the datasets to ensure equal representation (where the model sees samples from each downstream dataset the same number of times), performance deteriorates. This performance drop might be due to some tasks containing as few as 1020 total training samples, potentially causing overrepresentation of these smaller datasets during training.

C.10 SCS ablations

With SCS, we evaluate different repetition penalties and find that performance varies smoothly with α, reaching optimal results around $\alpha = 0.15$ (Table 13). For sampling efficiency, we introduce the batch size G, setting it to $100K$. This value is large enough for efficient medium scale sampling yet sufficiently small based on our empirical observation that while each sample had 1280 ($128M/100K$) opportunities to be selected, our most frequently sampled example appears only 56 times. To more directly validate our choice of G, Table 13 shows that using values of G of different order of magnitude has little effect on performance.

C.11 DataComp small scale results

For the small scale DataComp benchmark, we reuse the mixing model from the medium scale without retraining E-FLYT or M-FLYT. The only changes were re-tuning the repetition penalty α for the smaller dataset and reducing the sampling batch size G by a factor of 10 to $10K$, aligning with the dataset size reduction. Figure 4 shows that the smaller scale performs best with a higher repetition penalty, around $\alpha = 0.5$, compared to the medium scale which performs better with values between 0.15 to 0.25. A higher repetition penalty reduces the number of repetitions per example, which becomes more important for smaller datasets.

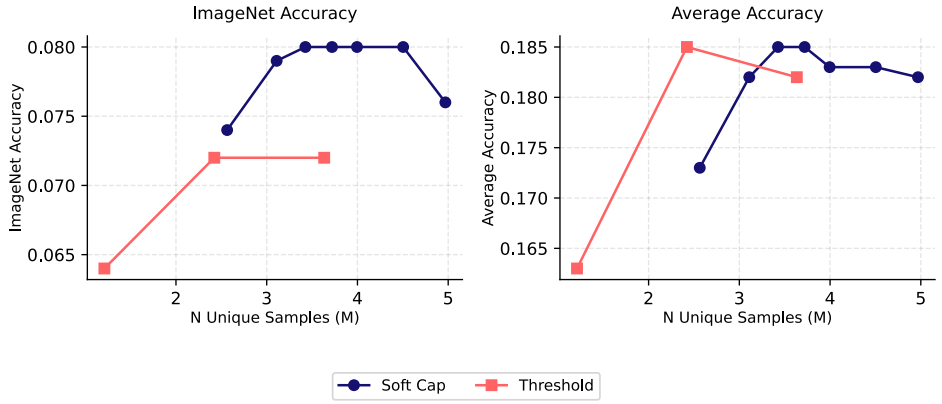

Figure 4: Comparing SCS to hard threshold filtering on the small scale DataComp benchmark. SCS was tested with α values from 0.15 to 1, and standard threshold with values from 10% to 30%

Table 14: Comparison between the results reported in the original papers and our reproduction using their provided UIDs. Some experiments were run using the complete DataComp medium scale dataset (128M) while our reproduction used the 119M examples we were able to download. Results marked with $*$ were performed by Wang et al. [49] on their 110M downloaded dataset; they also report partial results using the full 128M data pool. Our best results are listed below.

Filtering Method	Pool Size	ImageNet	ImageNet dist. shifts	VTAB	Retrieval	Average
s-CLIPLoss	110M	0.324*	0.274*	0.359*	0.263*	0.352*
	119M (Reprod.)	0.333	0.273	0.361	0.251	0.352
DFN	128M	0.371	0.298	0.388	0.288	0.373
	119M (Reprod.)	0.367	0.303	0.371	0.280	0.364
HYPE+DFN	116M	0.382	0.303	0.393	0.306	0.379
	119M (Reprod.)	0.372	0.304	0.374	0.287	0.368
DFN+s-CLIPLoss	128M, 110M*	0.375	0.309*	0.386*	0.281*	0.386
	119M (Reprod.)	0.371	0.304	0.390	0.285	0.378
HYPE+DFN+s-CLIPLoss	128M, 110M*	0.382	0.314*	0.385*	0.276*	0.388
	119M (Reprod.)	0.381	0.310	0.392	0.284	0.378
M-FLYT+SCS ($\alpha = 0.15$)	119M	0.401	0.311	0.396	0.292	0.377

D Reproduction comparison

Table 14 compares model we train using publicly available DataComp subsets to the numbers reported in the papers that published them. The results are generally similar, but not identical, with accuracy difference of up to 1%. The main driver of the difference appears to be the size of the actual pool used to train the model: we were able to download only 119M of the 128M examples in the DataComp medium scale pool, while the authors of the other papers had access to a more complete subset of the examples.

E Fixed M-FLYT baseline table

We include the baselines from Table 2 with correct configurations (subset size of $0.2 \cdot 119M$ instead of $0.2 \cdot 128M$) in Table 15.

Table 15: Baseline aggregation experiments from Table 2 with correct configuration (subset size of $0.2 \cdot 119M$ instead of $0.2 \cdot 128M$).

Baseline	ImageNet	Average
Sum	0.327	0.331
Standardized Sum	0.348	0.360
IN-weighted $r = 2$	0.350	0.364
IN-weighted $r = 4$	<u>0.354</u>	0.359
IN-weighted $r = 8$	0.353	<u>0.365</u>
IN-weighted $r = 16$	0.352	0.362
M-FLYT	**0.359**	**0.371**