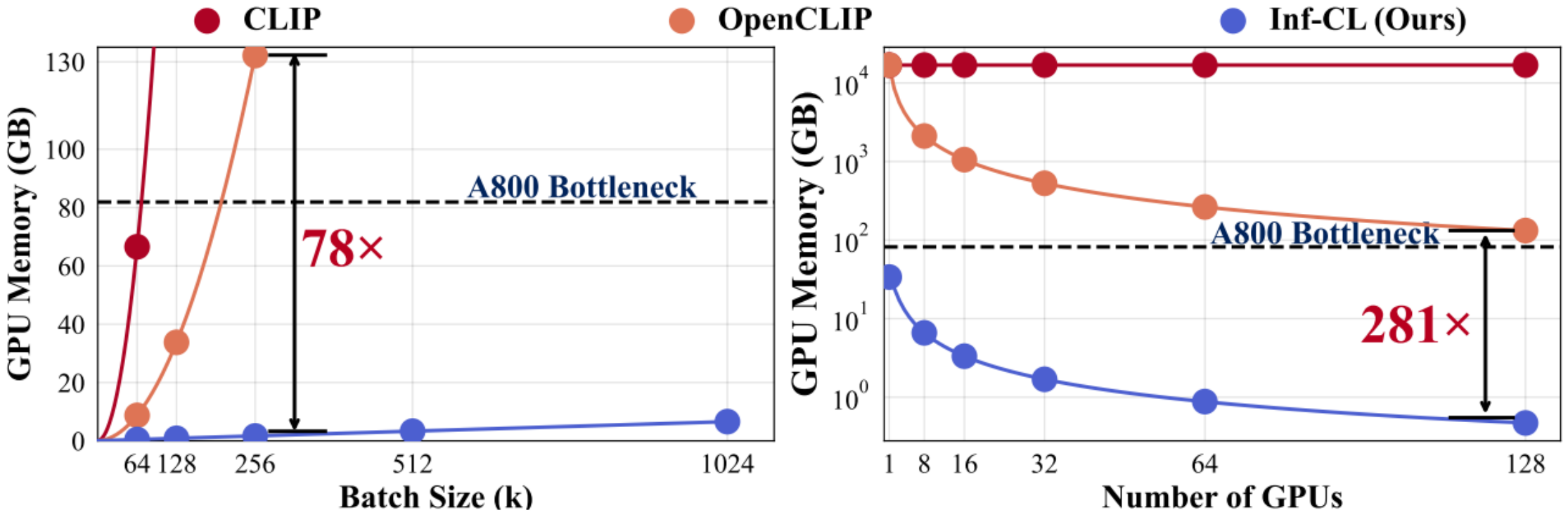
Breaking the Memory Barrier: Near Infinite Batch Size Scaling for Contrastive Loss

Zesen Cheng[2*], Hang Zhang[1,2*] ✉, Kehan Li[2*], Sicong Leng[2,3], Zhiqiang Hu[2], Fei Wu[1], Deli Zhao[2], Xin Li[2] ✉, Lidong Bing[2]
[1]Zhejiang University, [2]DAMO Academy, Alibaba Group, [3]Nanyang Technological University,
* Equal Contribution ✉ Corresponding Author

https://github.com/DAMO-NLP-SG/Inf-CLIP

Arxiv 2024

Result



- Left : Under the configuration of 8×A800 GPUs, the memory consumption of CLIP and OpenCLIP grows **quadratically**, while Inf-CL achieves **linear growth**. At a batch size of 256K, **Inf-CL reduces memory consumption by 78 times**.

- Right : At a batch size of 1024K, even with 128 GPUs, CLIP and OpenCLIP's memory will still be exhausted. In contrast, Inf-CL reduces the memory demand by 281 times.

Background

- Since 2020, contrastive learning has surged in popularity, with models like **SimCLR, MoCo, CLIP, SimCSE**, and DPR leveraging Contrastive Loss for image **self-supervised learning**, **cross-modal retrieval**, and **NLP** information retrieval.

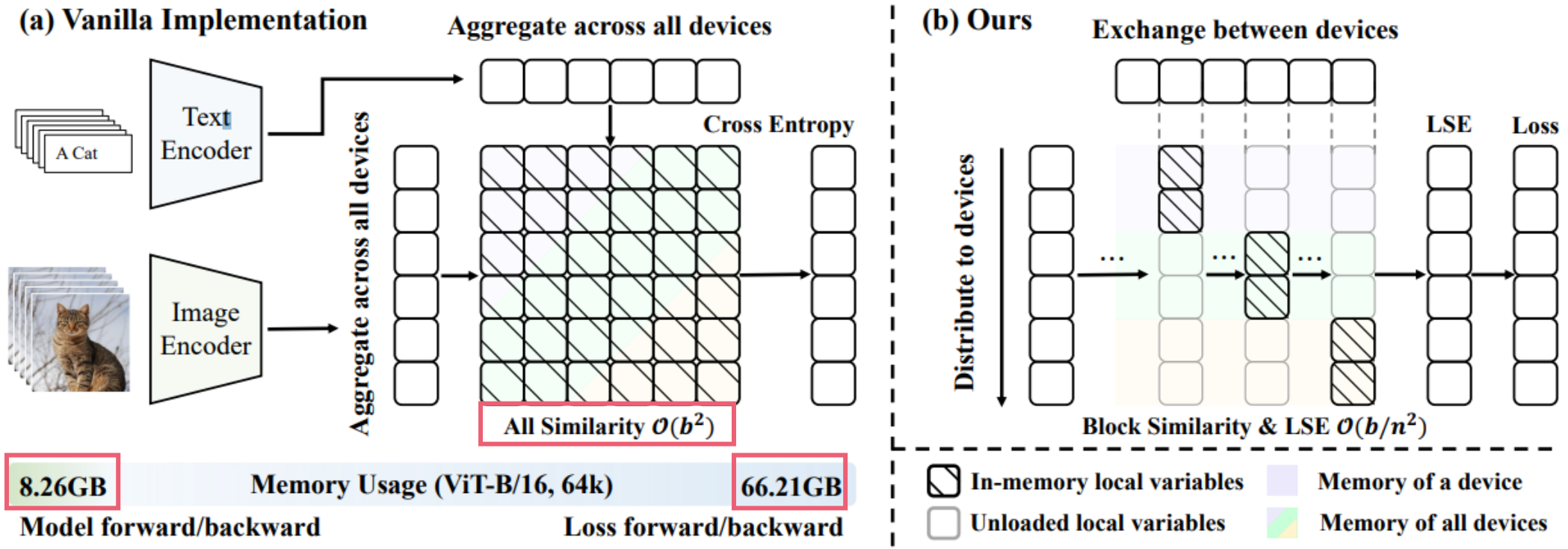
$$\mathcal{L}_I = -\frac{1}{b}\sum_{i=1}^{b}\log\frac{e^{x_{i,i}}}{\sum_{j=1}^{b}e^{x_{i,j}}}$$

$$x_{i,j} = I_i \cdot T_j$$

The cosine similarity between the i-th image and the j-th text.

- The **larger** the batch size, the **more negative samples** the model can encounter, thereby learning more discriminative features.
- In theory, the larger the batch size, the better the effect. **GPU memory overflow!**
- MoCo --> momenturm encoder & memory bank

Memory Limitation



=> Construct a **similarity matrix** in GPU Memory and compute loss using **Softmax** and **negative log-likelihood**.

=> The memory requirements for the similarity matrix and its normalization **results grow quadratically** with the batch size.

Bottleneck is mainly focused on loss calculation.

Contrastive Loss

$$\mathcal{L}_I = -\frac{1}{b}\sum_{i=1}^{b}\left(x_{i,i} - \log\sum_{j=1}^{b} e^{x_{i,j}}\right) = -\frac{1}{b}\sum_{i=1}^{b} x_{i,i} + \frac{1}{b}\sum_{i=1}^{b}\log\sum_{j=1}^{b} e^{x_{i,j}},$$

1. $\sum_{i=1}^{b} x_{i,i}$ => Calculate the similarity of all positive sample pairs and sum. $\mathcal{O}(b)$

2. $\sum_{i=1}^{b}\log\sum_{j=1}^{b} e^{x_{i,j}}$ => Calculate Log-Sum-Exp (LSE). The LSE of the similarities of all negative sample pairs. This part is calculated from the global similarity matrix.

Forward propagation.

$$l^{i,j} = LSE(X^{i,j})$$

$$\underbrace{\begin{bmatrix} X^{1,1} & \dots & X^{1,n_c} \\ \vdots & \ddots & \vdots \\ X^{n_r,1} & \dots & X^{n_r,n_c} \end{bmatrix}}_{\text{Tiled computation of } X} \rightarrow \underbrace{\begin{bmatrix} l^{1,1} & \dots & l^{1,n_c} \\ \vdots & \ddots & \vdots \\ l^{n_r,1} & \dots & l^{n_r,n_c} \end{bmatrix}}_{\text{Merged serially}} \rightarrow \begin{pmatrix} l^1 \\ \vdots \\ l^{n_r} \end{pmatrix} = l$$

- Only computing and storing parts of the similarity matrix instead of whole.
- Parallel processing across multiple GPUs, fitting parallel architectures.

Contrastive Loss

Overflow prevention strategy.

To prevent numerical instability and overflow during the merging process, the following numerically stable operation is performed:

$$l^i \leftarrow l^i + \log(1 + e^{l^{i,j} - l^i}),\ j = 1, \ldots, n_c,$$

initial value of l^i is 0.

During the computation of LSE(), direct exponentiation can lead to numerical overflow. To address this, using the following stabilized formulation:

$$l^{i,j} = \log \sum_k e^{X^{i,j}_{:,k}} = m^{i,j} + \log \sum_k e^{X^{i,j}_{:,k} - m^{i,j}},$$

$\boldsymbol{m}^{i,j} = \max_k \boldsymbol{X}^{i,j}_{:,k}$ is a vector.

Contrastive Loss

Backward propagation process.

Store only a subset of the similarity matrix at any given time, rather than the entire b × b matrix.

The gradients w.r.t. I and T are

$$\frac{\partial \mathcal{L}_I}{\partial \boldsymbol{I}_i} = \sum_j \frac{\partial \mathcal{L}_I}{\partial x_{i,j}} \cdot \frac{\partial x_{i,j}}{\partial \boldsymbol{I}_i}, \qquad \frac{\partial \mathcal{L}_I}{\partial \boldsymbol{T}_j} = \sum_i \frac{\partial \mathcal{L}_I}{\partial x_{i,j}} \cdot \frac{\partial x_{i,j}}{\partial \boldsymbol{T}_j}.$$

$$\Rightarrow \frac{\partial \mathcal{L}_I}{\partial \boldsymbol{I}_i} = -\frac{1}{b} \sum_j \left(\frac{\partial \mathcal{L}_I}{\partial x_{i,i}} \cdot \frac{\partial x_{i,i}}{\partial x_{i,j}} \cdot \frac{\partial x_{i,j}}{\partial \boldsymbol{I}_i} - \frac{\partial \mathcal{L}_I}{\partial l_i} \cdot \frac{\partial l_i}{\partial x_{i,j}} \cdot \frac{\partial x_{i,j}}{\partial \boldsymbol{I}_i} \right)$$

$$= -\frac{1}{b} \cdot \boldsymbol{T}_i + \frac{1}{b} \sum_j e^{x_{i,j} - l_i} \cdot \boldsymbol{T}_j. \qquad \Rightarrow \quad \boldsymbol{I}'_i \leftarrow \boldsymbol{I}'_i + e^{x_{i,j} - l_i} \cdot \boldsymbol{T}_j, \; j = 1, \ldots, n_c,$$

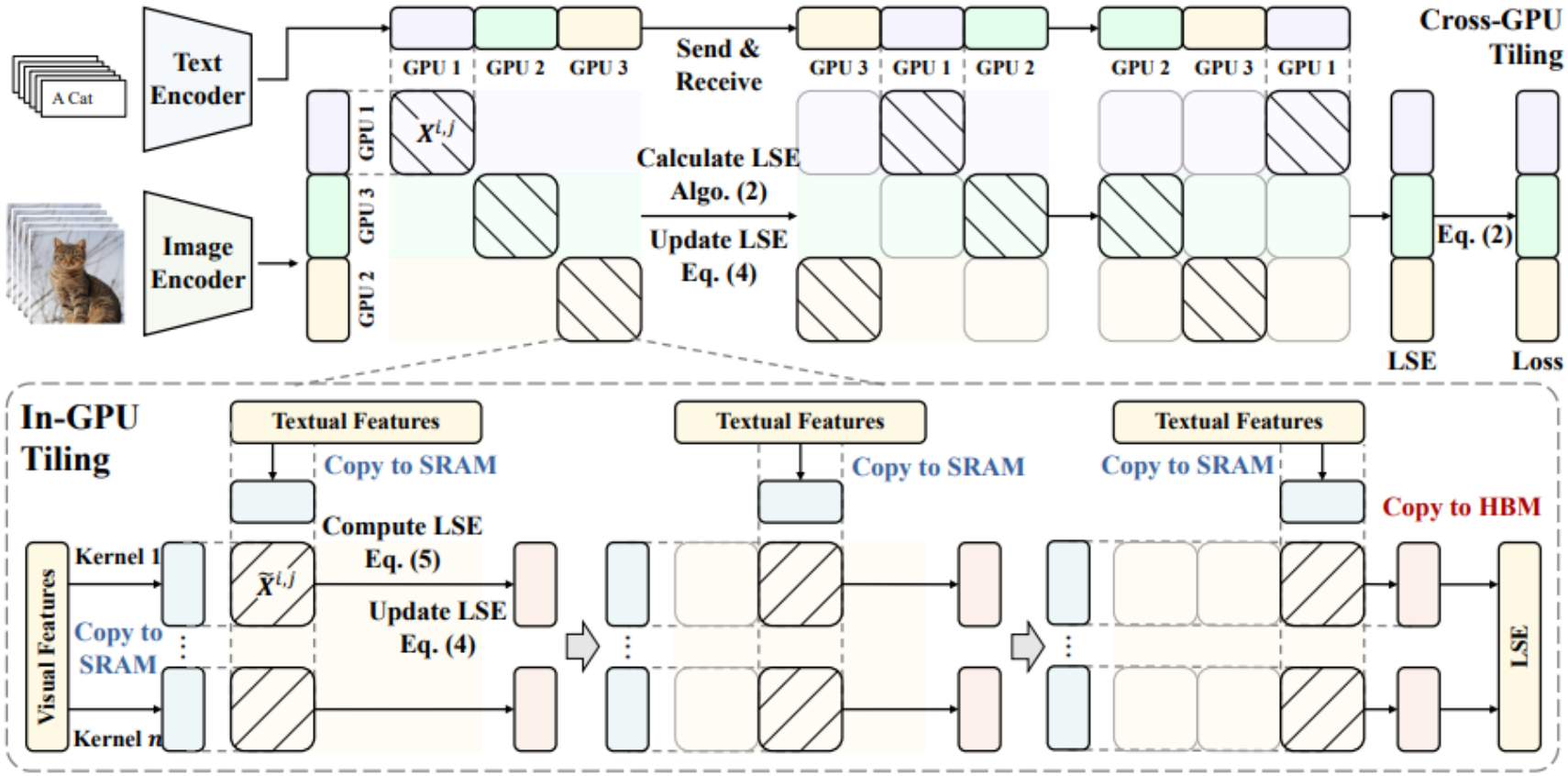
$$\Rightarrow \boxed{\frac{\partial \mathcal{L}_I}{\partial \boldsymbol{I}_i} = -\frac{1}{b} \cdot \boldsymbol{T}_i + \frac{1}{b} \boldsymbol{I}'_i,}$$

The same tile computation strategy :

1. During the forward, only a vector of size b is stored.
2. During the backward, gradients are accumulated tile by tile.

Contrastive Loss

Multi-Level Tiling



Top: In cross-GPU tiling, each GPU processes multiple rows, with asynchronous computation and column-wise communication to minimize costs.

Bottom: In in-GPU tiling, each GPU's calculations are divided into tiles, distributing row-wise computation across multiple CUDA cores. Row accumulations are combined into a single kernel to reduce OS I/O.

Memory Cost

Model	Loss (Peak) Memory Cost (GB)				
	32k	64k	128k	256k	1024k
8×A800 (≈ 8 × 80GB)					
CLIP	16.67 (46.40)	66.11 (77.94)	✗	✗	✗
OpenCLIP	2.27 (43.97)	8.63 (46.38)	33.64 (51.23)	✗	✗
Inf-CL	0.18 (44.20)	0.36 (46.63)	0.72 (51.46)	1.45 (61.13)	✗
Inf-CL*	0.18 (42.40)	0.36 (42.49)	0.72 (42.69)	1.45 (43.07)	6.53 (45.40)
32×A800 (≈ 32×80GB)					
CLIP	16.66 (42.85)	66.11 (75.52)	✗	✗	✗
OpenCLIP	0.71 (42.46)	2.45 (43.06)	8.98 (44.26)	34.35 (46.71)	✗
Inf-CL	0.05 (42.48)	0.09 (43.08)	0.18 (44.30)	0.35 (46.71)	1.44 (61.20)

Table 1: **Training Memory Cost Across Different Hardware and Batch Sizes**. Experiments utilize Data Parallelism with Automatic Mixed Precision for efficient distributed training. The baselines include the *Vanilla loss* (CLIP) and *Local loss* (OpenCLIP). To minimize memory consumption, Gradient Cache is adopted, with an accumulation batch size of 128. * indicates the use of the data offload strategy, which reduces memory usage by transferring only a small data batch from CPU to GPU during each accumulation step. ✗ denotes cases where the baseline exceeds the hardware memory limit for a given batch size, making training infeasible. Memory cost is evaluated using the ViT-L/14 architecture and the AdamW optimizer.

Memory Cost

Budget	Maximum Batch Size (Loss Memory Cost)			Improvement
	CLIP	OpenCLIP	Inf-CL	(Ours / Sota)
ViT-B/16				
8×A800	68k (74.39 GB)	172k (59.95 GB)	**800k** (3.01 GB)	4.65 (800k/172k)
32×A800	68k (74.39 GB)	360k (66.29 GB)	**3456k** (3.27 GB)	9.60 (3456k/360k)
ViT-L/14				
8×A800	64k (66.11 GB)	152k (47.23 GB)	**448k** (2.52 GB)	2.94 (448k/152k)
32×A800	64k (66.11 GB)	352k (64.13 GB)	**2048k** (2.89 GB)	5.82 (2048k/256k)
ViT-L/14 w/ data offload				
8×A800	64k (66.11 GB)	184k (69.10 GB)	**4096k** (26.12 GB)	22.26 (4096k/184k)
32×A800	64k (66.11 GB)	368k (64.13 GB)	**12288k** (19.59 GB)	33.39 (12288k/368k)

Table 2: **Maximum batch size** for model training using different hardware and contrastive loss methods. The training setting of this experiment is aligned with Table 1.

Experiment

Speed

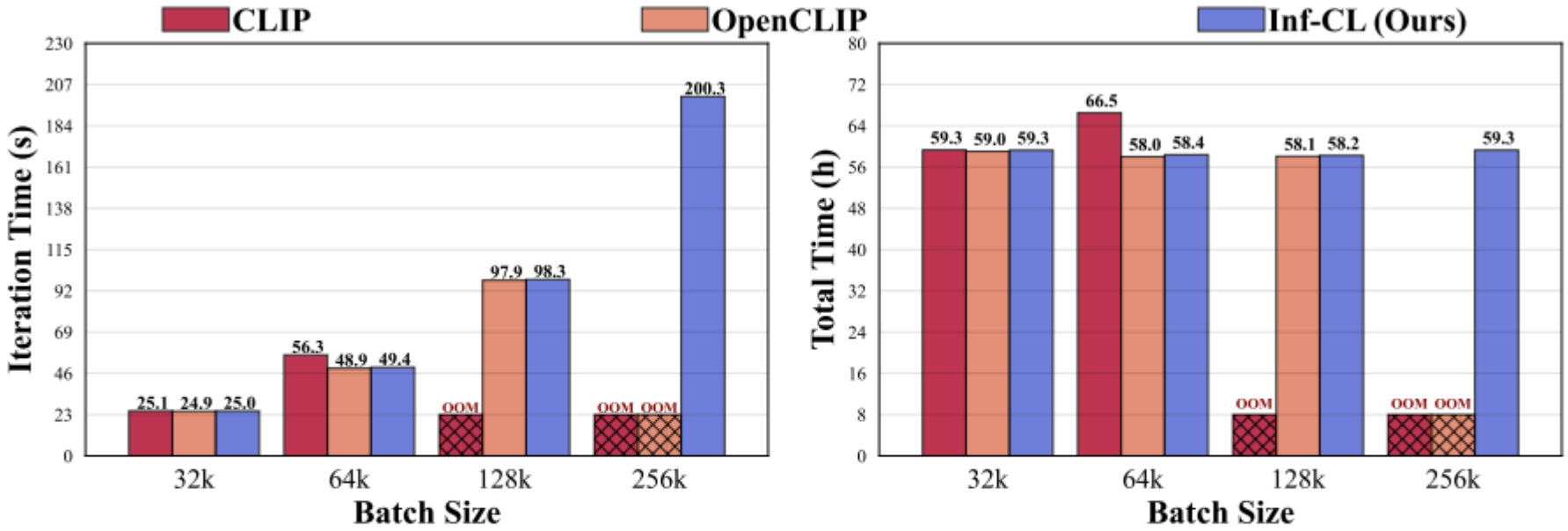


Figure 4: **Training Speed of ViT-L/14 CLIP on 8×A800 for Varying Batch Sizes.** The left figure shows the time per iteration step, while the right displays the time per epoch. Loss calculation contributes minimally to the total iteration time, making Inf-CL's iteration time comparable to previous methods. Furthermore, the iteration time of **Inf-CL** scales linearly with batch size, leading to a stable training duration of approximately 59 hours per epoch.

Performance

Method (Batch Size)	ImageNet				MSCOCO R@1	
	Validation	v2	ObjectNet	OOD	I→T	T→I
Vanilla (64K)	74.74	**65.30**	46.31	66.13	25.71	44.31
OpenCLIP (64K)	74.86	65.22	46.29	66.75	25.98	44.02
Inf-CL (64K)	74.93	65.27	46.13	66.77	**26.01**	43.95
Inf-CL (256K)	**75.12**	65.12	**46.44**	**67.15**	25.90	**44.61**
Inf-CL (1024K)	73.58	63.87	44.55	64.60	24.53	41.58

Table 3: **Performance Verification.** The training strategies is consistent with Table 2. We choose ViT-B/16 as the model architecture and adopt LiT strategy like Table 4. We evaluate zero-shot top-1 classification accuracy on several data sets, e.g., ImageNet-Validation Deng et al. (2009), ImageNet-v2 (Recht et al., 2019), ObjectNet (Barbu et al., 2019) and ImageNet-OOD (Hendrycks et al., 2021). We also evaluate zero-shot image-text top-1 retrieval accuracy on MSCOCO (Chen et al., 2015).

Performance

Method (Batch Size)	ImageNet				MSCOCO R@1	
	Validation	v2	ObjectNet	OOD	I→T	T→I
Vanilla (64K)	74.74	**65.30**	46.31	66.13	25.71	44.31
OpenCLIP (64K)	74.86	65.22	46.29	66.75	25.98	44.02
Inf-CL (64K)	74.93	65.27	46.13	66.77	**26.01**	43.95
Inf-CL (256K)	**75.12**	65.12	**46.44**	**67.15**	25.90	**44.61**
Inf-CL (1024K)	73.58	63.87	44.55	64.60	24.53	41.58

Table 3: **Performance Verification.** The training strategies is consistent with Table 2. We choose ViT-B/16 as the model architecture and adopt LiT strategy like Table 4. We evaluate zero-shot top-1 classification accuracy on several data sets, e.g., ImageNet-Validation Deng et al. (2009), ImageNet-v2 (Recht et al., 2019), ObjectNet (Barbu et al., 2019) and ImageNet-OOD (Hendrycks et al., 2021). We also evaluate zero-shot image-text top-1 retrieval accuracy on MSCOCO (Chen et al., 2015).

Experiment

Ablation Study

Cross-GPU	In-GPU	Data Memory	Loss Complexity	Loss Memory	Backbone Memory	Peak Memory	ImageNet
(Vanilla)		1.96	$\mathcal{O}(b^2)$	66.21	8.26	69.24	74.82
(OpenCLIP)		1.96	$\mathcal{O}(b^2/n)$	16.96	8.26	20.79	74.86
✔		1.96	$\mathcal{O}(b^2/n^2)$	4.81	8.26	12.30	74.78
✔	✔	1.96	$\mathcal{O}(b/n^2)$	0.81	8.26	12.30	74.93

Table 4: **Ablation Study of Multi-level Tiling Strategy.** The training strategies is consistent with Table 2, using the ViT-B/16 architecture. To reduce memory consumption and expedite experimentation, we freeze the image encoder and load pretrained weights as done in LiT. The global batch size is fixed at 64k with an accumulation batch size of 256 per GPU. These experiments are conducted on 4×A800 (80G) GPUs. "Complexity" denotes the space complexity of loss calculation. b denotes batch size, while n denotes the number of GPUs.

Thanks