This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Breaking the Memory Barrier of Contrastive Loss via Tile-Based Strategy

Zesen Cheng[1*] Hang Zhang[1,2*] ✉ Kehan Li[1*] Sicong Leng[1] Zhiqiang Hu[1]
Fei Wu[2] Deli Zhao[1] Xin Li[1] ✉ Lidong Bing[1,3]

[1] Alibaba Group, Hangzhou, China [2] Zhejiang University, Hangzhou, China
[3] Shanda AI Research Institute

{chengzesen.czs,hang.zh,likehan.lkh,xinting.lx}@alibaba-inc.com

Abstract

Contrastive loss is a powerful approach for representation learning, where larger batch sizes enhance performance by providing more negative samples to better distinguish between similar and dissimilar data. However, the full instantiation of the similarity matrix demands substantial GPU memory, making large batch training highly resource-intensive. To address this, we propose a tile-based computation strategy that partitions the contrastive loss calculation into small blocks, avoiding full materialization of the similarity matrix. Additionally, we introduce a multi-level tiling implementation to leverage the hierarchical structure of distributed systems, using ring-based communication at the GPU level to optimize synchronization and fused kernels at the CUDA core level to reduce I/O overhead. Experimental results show that the proposed method significantly reduces GPU memory usage in contrastive loss. For instance, it enables contrastive training of a CLIP-ViT-L/14 model with a batch size of 4M using only 8 A800 80GB GPUs, without sacrificing accuracy. Compared to state-of-the-art memory-efficient solutions, it achieves a ***two-order-of-magnitude*** *reduction in memory while maintaining comparable speed. The code will be made publicly available.*[1]

1. Introduction

Contrastive learning serves as a foundational technique across various applications, such as multi-modality retrieval [12, 21, 25], self-supervised learning [3, 11, 14], and dense text retrieval [33]. It learns an embedding space in which similar data pairs stay close while dissimilar ones are far apart [13, 23, 34]. Large batch sizes are critical to the success of contrastive learning due to their reliance on in-batch negatives [3, 25]. Specifically, larger batches provide a diverse set of negative samples, enhancing the model's ability to learn discriminative representations [24].

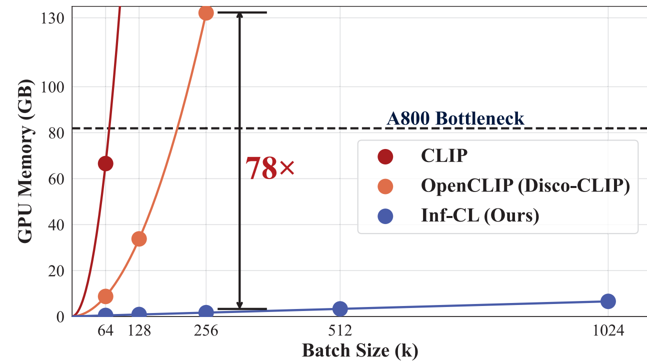

Figure 1. **GPU memory usage comparison** between **Inf-CL** and previous methods. The dashed line marks the common GPU memory limit. Results exceeding the bottleneck are estimated by curve fitting. With 8×A800, CLIP and OpenCLIP's memory consumption increases quadratically, while Inf-CL achieves linear growth, reducing memory costs by **78×** at a batch size of 256k.

Despite the potential benefits of large batch sizes in contrastive learning, training with large batches remains a significant challenge under constrained GPU memory. The memory required to compute and store image-text similarity matrices (Figure 1) grows quadratically with batch size, creating substantial barriers to scalability. This issue severely limits the practical adoption of large batch sizes, even when advanced hardware is available. To address this challenge, several methods have been proposed to reduce memory usage. Gradient-Cache [10] reduces memory usage by decoupling model and loss computations, but the memory cost of the loss still poses a significant bottleneck. OpenCLIP [16] and DisCo-CLIP [6] enhance efficiency by distributing contrastive loss computation across n GPUs, reducing memory consumption by a factor of n. However, even with these advancements, the memory cost of calculating contrastive loss remains substantial, requiring necessitating numerous expensive high-memory GPUs to support training (Figure 1).

In this paper, we introduce **Inf-CL**, a novel approach

* Equal contribution ✉ Corresponding Author
[1] github.com/DAMO-NLP-SG/Inf-CLIP

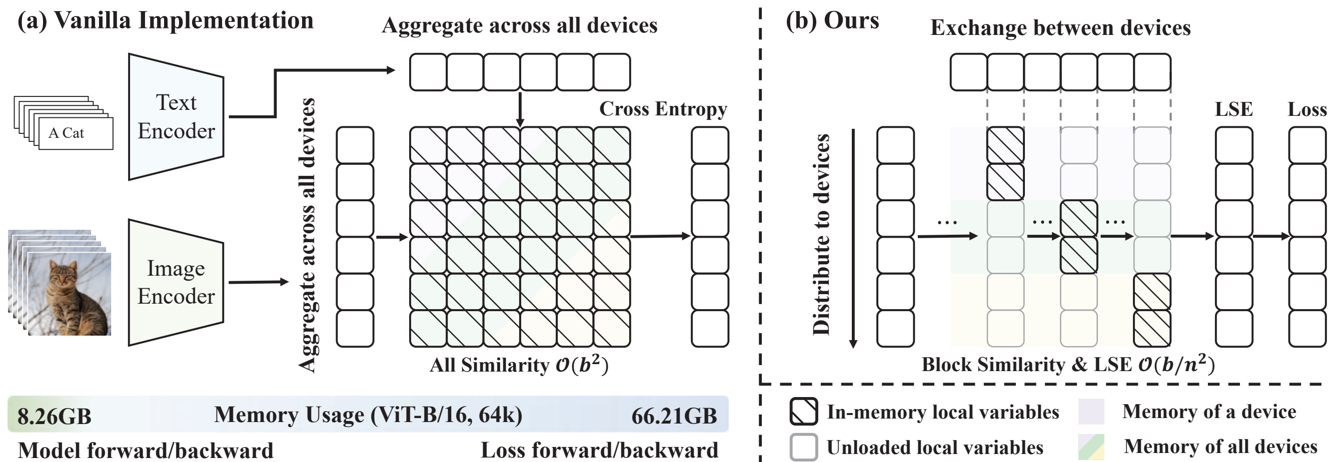

Figure 2. (a) **Vanilla implementation of contrastive loss** gathers features to all devices to calculate all similarity simultaneously, where the similarity with squared complexity are repeatedly stored in all devices, causing huge memory costs for loss calculation when batch size increases. (b) **Our Inf-CL** significant decreases the memory cost by serial and distributed tile-wise computation.

to mitigate the quadratic memory cost in contrastive learning, which is caused by the full instantiation of the similarity matrix for log-sum-exp (LSE) computation. Instead of storing the entire matrix, **Inf-CL** partitions the LSE calculation into smaller, sequentially computed tiles, leveraging the cumulative property of LSE. This confines memory usage to the tile size and the number of parallel tiles, allowing for a trade-off between memory and computational efficiency. To enhance practical efficiency, we propose a multi-level tiling strategy. At a coarse-grained level, image and text batches are distributed across multiple GPUs, with each GPU performing serial LSE computations on multiple rows. As computations proceed, asynchronous column-wise data exchange minimizes communication overhead, as illustrated in Figure 2(b). At a fine-grained level, row-wise computations are parallelized across CUDA cores within each GPU, consolidating iterations into a single kernel to reduce I/O overhead. The multi-level tiling strategy is crucial to achieving practical scalability and efficiency, balancing memory reduction with computation speed.

We evaluate **Inf-CL** on the image-text contrastive learning task. As shown in Figure 1, **Inf-CL** reduces space complexity from quadratic (e.g., $\mathcal{O}(b^2)$ for CLIP, $\mathcal{O}(b^2/n)$ for OpenCLIP) to linear ($\mathcal{O}(b/n^2)$ for **Inf-CL**), where b and n are the batch size and the number of GPUs. This substantial reduction in memory usage allows training with large batch sizes when resources are limited. For instance, computing the contrastive loss with a batch size of 256k on 8 GPUs requires only 1.63 GB of memory per GPU—an impressive **78×** improvement over previous methods. Moreover, **Inf-CL** maintains precision consistent with existing approaches. In terms of computation time, **Inf-CL** matches the performance of prior methods, taking approximately 59 hours to process a 64k batch size on 8 A800 GPUs.

2. Related Work

Contrastive Learning: The core idea of contrastive learning is to learn better representations by distinguishing between positive and negative pairs of samples [4, 31]. This approach demonstrates strong effectiveness across diverse tasks, as the nature of the paired samples varies depending on the specific application. In image foundation models, such as SimCLR [3] and MoCo [14], positive pairs are created by augmenting the same image in different ways. For cross-modal retrieval, as exemplified by CLIP [25] and ALIGN [17], the positive pairs consist of aligned image and text samples. Similarly, for dense text retrieval [18, 33, 39], the positive pairs are composed of query and document pairs. Several works improve contrastive learning performance by enhancing dataset quality, modifying the loss function, or refining negative sample selection [32, 38, 40]. Moreover, several studies, both empirical and theoretical, have demonstrated from various perspectives that larger batch sizes contribute to learning better representations [2, 27]. Due to the quadratic growth of memory usage with batch size in classical contrastive loss, most existing studies have stopped scaling their batch sizes to 128k, even when leveraging hundreds of GPUs [17, 25, 35].

Memory-efficient Training: As deep learning models continue to grow in size and complexity, the demand for computational resources, particularly GPU memory, has increased significantly. Techniques such as Gradient Checkpointing [30] recompute activations during backpropagation to save memory at the expense of additional computation. Flash Attention [7] reduces memory overhead by computing attention in blocks without storing large intermediate states. Ring Attention [20] distributes long sequence activations across multiple devices, overlapping computation and communication to train sequences far longer than previ-

ous methods. For contrastive learning, GradCache [10] and BASIC [24] introduce a gradient caching technique that decouples backpropagation between contrastive loss and the encoder, which reduces memory usage in the model by accumulating gradients per batch. OpenCLIP [16] and DisCo-CLIP [6] reduce memory usage by distributing the computation of contrastive loss across multiple GPUs.

3. Preliminaries

3.1. Distributed training system

Cross-GPU Communication: Modern deep learning models are typically trained using multiple GPUs. However, communication overhead between GPUs can limit performance. Techniques like hierarchical all-reduce and ring-based communication alleviate such overhead by optimizing synchronization between GPUs [20]. Blockwise parallelism, as employed in methods like ring attention [20], further improves efficiency by overlapping computation and communication.

GPU Memory and Execution: The performance of modern deep learning models relies heavily on hardware resources, particularly GPU memory and execution capabilities. GPUs, like A100s, typically have two different types of memory: HBM (High Bandwidth Memory) and SRAM (Static Random Access Memory). HBM serves as the primary memory with a capacity of up to 80GB. In contrast, SRAM is much smaller (usually measured in megabytes) but offers a significantly faster access speed, acting as a vital cache for frequently accessed data and enabling rapid computations. Techniques like FlashAttention [7] show that fine-grained control over the memory access of HBM and the fuse the operations can achieve faster training and less memory usage.

3.2. Vanilla Implementation of Contrastive Loss

In contrastive loss, the objective is to learn an embedding space where similar samples (positive pairs) are pulled closer, while dissimilar samples (negative pairs) are pushed away. A typical implementation, exemplified by CLIP [25], is depicted in Figure 2. The image and text encoders are trained with contrastive loss after extracting features. For brevity, we only discuss image-to-text contrastive loss as an example in the following sections, since the implementation of text-to-image loss is symmetric. Specifically, given a batch size of b, the in-batch c-dimensional visual feature $\boldsymbol{I} \in \mathbb{R}^{b \times c}$, and textual feature $\boldsymbol{T} \in \mathbb{R}^{b \times c}$, the contrastive loss is defined as

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

where $x_{i,j} = \boldsymbol{I}_i \cdot \boldsymbol{T}_j$ is the scaled cosine similarity between the i-th image and j-th text, and $x_{i,i}$ represents the positive pair. Here, we omitted the temperature factor for simplicity.

The vanilla implementation first computes the similarity matrix $\boldsymbol{X} \in \mathbb{R}^{b \times b} = \boldsymbol{I} \cdot \boldsymbol{T}'$ and stores it in high-bandwidth memory (HBM). Afterward, softmax normalization followed by the calculation of negative log-likelihood is applied to the similarity matrix. The memory required to store $\boldsymbol{X}$ and its normalized results scales as $\mathcal{O}(b^2)$, which can occupy a substantial amount of GPU memory when b is large. Figure 2 gives an example of training ViT-B/16 with a batch size of 64k, using model memory optimization techniques such as Gradient Cache [10, 24]. As can be seen, the GPU memory footprint of the model itself is only 5.24GB while the loss calculation still requires 66GB. This indicates that, with batch size scaling, the memory bottleneck during training lies in the loss calculation. Although large batch sizes are necessary for improving model performance [2, 27], the traditional implementation struggles to support them due to excessive memory consumption in the loss calculation.

4. Method

4.1. Tile-wise Contrastive Learning

As discussed in Section 3.2, the root cause of the quadratic memory growth in the vanilla implementation is the full materialization of the similarity matrix $\boldsymbol{X}$. To decrease the memory cost, we first decompose the operations related to $\boldsymbol{X}$ from the loss function:

$$\begin{aligned} \mathcal{L}_I &= -\frac{1}{b} \sum_{i=1}^{b} (x_{i,i} - \log \sum_{j=1}^{b} e^{x_{i,j}}) \\ &= -\frac{1}{b} \sum_{i=1}^{b} x_{i,i} + \frac{1}{b} \sum_{i=1}^{b} \log \sum_{j=1}^{b} e^{x_{i,j}}, \end{aligned} \tag{2}$$

where the spatial complexity of the first part is $\mathcal{O}(b)$, and $\mathcal{O}(b^2)$ for the second log-sum-exp (LSE) part. Based on this formulation, we introduce a tile-wise contrastive loss implementation that avoids the full instantiation of $\boldsymbol{X}$ by iterative accumulation between tiles. The following sections provide detailed formulation for forward and backward.

Tile-Wise Forward. To reduce the dependency on storing $\boldsymbol{X}$ entirely, we adopt a tile-wise approach for calculating $\boldsymbol{l}$. The process is show as below:

$$\underbrace{\begin{bmatrix} \boldsymbol{X}^{1,1} & \cdots & \boldsymbol{X}^{1,n_c} \\ \vdots & \ddots & \vdots \\ \boldsymbol{X}^{n_r,1} & \cdots & \boldsymbol{X}^{n_r,n_c} \end{bmatrix}}_{\text{Tiled computation of } \boldsymbol{X}} \rightarrow \underbrace{\begin{bmatrix} \boldsymbol{l}^{1,1} & \cdots & \boldsymbol{l}^{1,n_c} \\ \vdots & \ddots & \vdots \\ \boldsymbol{l}^{n_r,1} & \cdots & \boldsymbol{l}^{n_r,n_c} \end{bmatrix}}_{\text{Merged serially via Eq. 4}} \tag{3}$$

where n_r and n_c represent the number of tiles along the rows and columns, respectively. The computation proceeds

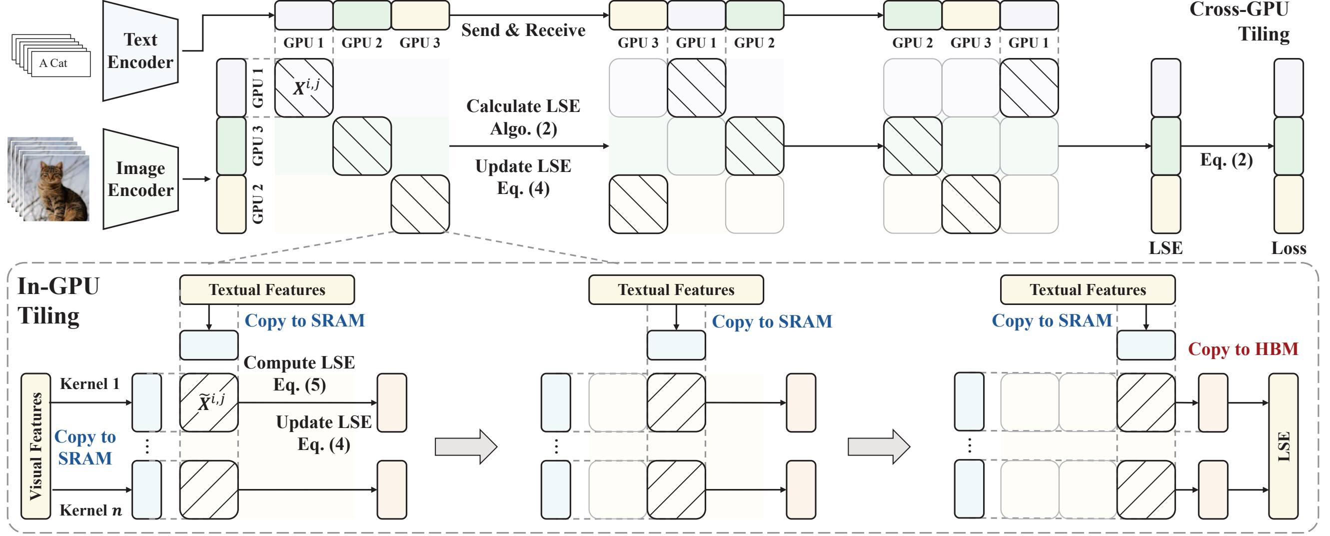

Figure 3. **Multi-level tiling strategy. Top**: for cross-GPU tiling, each GPU is assigned with multiple rows. The computation and the column-wise communication are performed asynchronously to reduce the cost. **Bottom**: for in-GPU tiling, the calculations in each GPU are further divided into tiles and the row-wise calculation is distributed to multiple CUDA cores. The accumulative operations of each row are merged into one kernel for reducing I/O times between SRAM and HBM.

by dividing $\boldsymbol{X}$ into multiple tiles, denoted as $\boldsymbol{X}^{i,j}$, and then calculating the intermediate LSE values $\boldsymbol{l}^{i,j} = \mathrm{LSE}(\boldsymbol{X}^{i,j})$ within each tile. The resulting LSE values from each column of tiles are then merged serially along the rows to obtain the final global LSE vector $\boldsymbol{l}$.

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

$$\boldsymbol{l}^i \leftarrow \boldsymbol{l}^i + \log(1 + e^{\boldsymbol{l}^{i,j} - \boldsymbol{l}^i}),\ j = 1, \ldots, n_c, \tag{4}$$

where the initial value of $\boldsymbol{l}^i$ is 0. In each iteration, the intermediate value $\boldsymbol{l}^{i,j}$ is merged with $\boldsymbol{l}^i$, and after processing all n_c tiles, the global LSE vector $\boldsymbol{l}$ is obtained.

During the computation of $\mathrm{LSE}(\boldsymbol{X}^{i,j})$, direct exponentiation can lead to numerical overflow. To address this, we compute $\boldsymbol{l}^{i,j}$ using the following stabilized formulation:

$$\boldsymbol{l}^{i,j} = \log \sum_k e^{\boldsymbol{X}^{i,j}_{:,k}} = \boldsymbol{m}^{i,j} + \log \sum_k e^{\boldsymbol{X}^{i,j}_{:,k} - \boldsymbol{m}^{i,j}}, \tag{5}$$

This vector acts as a normalization factor, ensuring that the values inside the exponential function remain numerically stable. where $\boldsymbol{m}^{i,j} = \max_k \boldsymbol{X}^{i,j}_{:,k}$ is the row-wise maximum, which acts as a normalization factor for ensuring the numerically stable of the exponential function.

This tile-wise approach significantly reduces the memory requirement by allowing each GPU to compute and store only a subset of the similarity matrix at any given time, rather than the entire $b \times b$ matrix. Additionally, this method facilitates scaling to larger batch sizes by enabling parallel computation of the tiles on multiple GPUs or across different nodes in a distributed system.

Tile-Wise Backward. According to the chain rule, the gradients *w.r.t.* $\boldsymbol{I}_i$ and $\boldsymbol{T}_j$ are

$$\begin{aligned} \frac{\partial \mathcal{L}_I}{\partial \boldsymbol{I}_i} &= \sum_j \frac{\partial \mathcal{L}_I}{\partial x_{i,j}} \cdot \frac{\partial x_{i,j}}{\partial \boldsymbol{I}_i}, \\ \frac{\partial \mathcal{L}_I}{\partial \boldsymbol{T}_j} &= \sum_i \frac{\partial \mathcal{L}_I}{\partial x_{i,j}} \cdot \frac{\partial x_{i,j}}{\partial \boldsymbol{T}_j}. \end{aligned} \tag{6}$$

Taking the gradients *w.r.t.* $\boldsymbol{I}_i$ as an example, according to Equation 2, the complete formulation is

$$\begin{aligned} \frac{\partial \mathcal{L}_I}{\partial \boldsymbol{I}_i} &= -\frac{1}{b} \cdot \boldsymbol{T}_i + \frac{1}{b} \sum_j \frac{\partial \mathcal{L}_I}{\partial l_i} \cdot \frac{\partial l_i}{\partial x_{i,j}} \cdot \frac{\partial x_{i,j}}{\partial \boldsymbol{I}_i} \\ &= -\frac{1}{b} \cdot \boldsymbol{T}_i + \frac{1}{b} \sum_j e^{x_{i,j} - l_i} \cdot \boldsymbol{T}_j. \end{aligned} \tag{7}$$

From the formula, it can be seen that the second term requires the similarities $x_{i,j}$ with $\mathcal{O}(b^2)$ memory for whether storing it in the forward process or computing it directly in the backward process. To tackle this, we apply the similar tile-based method as the forward process to compute the gradient. Specifically, we first store $\boldsymbol{l}$, which has only b elements during forward propagation, and calculate the gradient *w.r.t* $\boldsymbol{I}_i$ by iterative accumulation in multiple tiles:

$$\begin{aligned} \boldsymbol{I}'_i &\leftarrow \boldsymbol{I}'_i + e^{x_{i,j} - l_i} \cdot \boldsymbol{T}_j,\ j = 1, \ldots, n_c, \\ \frac{\partial \mathcal{L}_I}{\partial \boldsymbol{I}_i} &= -\frac{1}{b} \cdot \boldsymbol{T}_i + \frac{1}{b} \boldsymbol{I}'_i, \end{aligned} \tag{8}$$

where $\boldsymbol{I}'_i$ is a temporary variable for accumulation. The detailed algorithm is shown in Appendix.

Algorithm 1 Forward Process of Multi-level Tile-Wise Global LSE Calculation

Require: Number of GPUs n, in-memory visual features and textual features $\boldsymbol{I}^i, \boldsymbol{T}^i \in \mathbb{R}^{b_s \times c}$ for each GPU.

1: **for** $counter$ = 1 **to** n **do**
2: **Update LSE:**
3: Each GPU computes the local LSE vector via Algorithm 2 with in-memory features $\boldsymbol{I}^i, \boldsymbol{T}^i$.
4: Each GPU updates the LSE vector via Equation 4.
5: **Asynchronously Communication:**
6: Each GPU sends the in-memory textual feature to the next GPU in the ring.
7: Each GPU receives the textual feature from the previous GPU in the ring.
8: **end for**
9: Return the final LSE vector $\boldsymbol{l}_i$ for each GPU .

Algorithm 2 Forward Process of In-GPU LSE Calculation

Require: Visual features: $\tilde{\boldsymbol{I}} \in \mathbb{R}^{b_s \times c}$ and textual features: $\tilde{\boldsymbol{T}} \in \mathbb{R}^{b_s \times c}$, the row-wise and column-wise size of a tile: t_r and t_c.

1: Divide $\tilde{\boldsymbol{I}}$ into $\tilde{\boldsymbol{I}}^i$, where $i = 1, 2, \ldots, \tilde{n}_r$.
2: Divide $\tilde{\boldsymbol{T}}$ into $\tilde{\boldsymbol{T}}^j$, where $j = 1, 2, \ldots, \tilde{n}_c$.
3: **parallel for** each $\tilde{\boldsymbol{I}}^i$ **do**
4: Load $\tilde{\boldsymbol{I}}^i$ from HBM to on-chip SRAM.
5: Initialize $\tilde{\boldsymbol{l}}^i = \boldsymbol{0} \in \mathbb{R}^{t_r}$.
6: **for** j = 1 **to** $\tilde{n}_r$ **do**
7: Load $\tilde{\boldsymbol{T}}_j$ from HBM to on-chip SRAM.
8: On chip, compute $\tilde{\boldsymbol{X}}^{i,j} = \tilde{\boldsymbol{I}}^i \cdot \tilde{\boldsymbol{T}}^{j\prime} \in \mathbb{R}^{t_r \times t_c}$.
9: On chip, calculate $\tilde{\boldsymbol{l}}^{i,j}$ based on Equation 5.
10: On chip, update LSE $\tilde{\boldsymbol{l}}^i$ based on Equation 4.
11: **end for**
12: Write $\tilde{\boldsymbol{l}}^i$ to HBM.
13: **end parallel for**
14: Return $\tilde{\boldsymbol{l}}$.

4.2. Multi-Level Tiling

The scaling of batch size is usually accompanied by the scaling of the number of GPUs. In order to fully utilize the parallelism between multiple GPUs while exploiting partially serial computation on a single GPU to reduce the memory cost, we propose a multi-level tiling method that distributes the above LSE calculation to coarse-grained cross-GPU tiles and fine-grained in-GPU tiles.

Cross-GPU Tile. As shown in Algorithm 1, in data parallel training with n GPUs, the i-th GPU first processes a portion of images and texts to visual features $\boldsymbol{I}^i \in \mathbb{R}^{b_s \times c}$ and textual features $\boldsymbol{T}^i \in \mathbb{R}^{b_s \times c}$, where $b_s = b/n$ is the batch size in one GPU. Then for the calculation of the contrastive loss, we distribute computations of different rows to different GPUs and synchronize the columns between GPUs step-by-step, considering the row-wise characteristic. Specifically, the i-th GPU is responsible for calculating $\boldsymbol{X}^{i,:}$ and the corresponding $\boldsymbol{l}^i$. For memory considerations, based on the tiling strategy described in Section 4.1 where only one tile $\boldsymbol{X}^{i,j}$ is computed at a time, $\boldsymbol{X}^{i,:}$ is further divided into $\boldsymbol{X}^{i,j}$ for n step to calculate $\boldsymbol{l}^i$ following Equation 4, where the local LSE $\boldsymbol{l}^{i,j}$ is calculated by in-gpu tiling as described in the next part.

Moreover, since the computation of $\boldsymbol{X}^{i,j}$ while $i \neq j$ requires the textual feature $\boldsymbol{T}^j$ stored in other GPUs, additional communication overhead is inevitable, especially as the number of GPUs grows. In order to reduce or even eliminate the communication overhead, we associate all GPUs with a ring topology, based on the idea of overlapping communication time and computation time overlap as much as possible. Concretely, starting with $\boldsymbol{T}^i$, each GPU process sends the current textual features to the next process and receives the textual features from the previous process using the ring topology while computing Equation 4. In this way, the communication time cost is negligible when it is greater than the computation time overhead.

In-GPU Tile. With the cross-GPU tiling technique, the memory complexity becomes $\mathcal{O}(b_s^2)$ for directly storing $\boldsymbol{X}^{i,j}$ where $b_s = b/n$. Since the number of GPU n is somehow limited, we further introduce in-GPU tiling to reduce the $\mathcal{O}(b_s^2)$ memory cost to $\mathcal{O}(b_s)$. Specifically, we first split $\tilde{\boldsymbol{X}} = \boldsymbol{X}^{i,j}$ into tiles:

$$\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{X}}^{i,j}],\ i = 1, \ldots, \tilde{n}_r,\ j = 1, \ldots, \tilde{n}_c, \tag{9}$$

where $\tilde{n}_r = \lceil b/t_r \rceil$ and $\tilde{n}_c = \lceil b/t_c \rceil$ and t_r and t_c is the row-wise and column-wise size of a tile. For implementation, we distribute rows to multiple CUDA cores to make full use of the parallel computing power of GPU, and serial process the row-wise tiles in each kernel by applying Equation 5 and Equation 4 to $\tilde{\boldsymbol{X}}^{i,j}$, as shown in Algorithm 2.

The iterative computation requires multiple memory access for variable $\boldsymbol{l}^i$. To avoid expensive I/O from HBM to SRAM, we fuse the row-wise iterative calculation into one kernel. Specifically, $\boldsymbol{l}^i$ and $\tilde{\boldsymbol{X}}^{i,j}$ are allocated in SRAM. In this way, the image features are loaded to SRAM only once at beginning, and $\tilde{\boldsymbol{l}}^i$ is written to HBM only once in the end, as shown in Figure 3.

Model	Loss (Peak) Memory Cost (GB)				
	32k	64k	128k	256k	1024k
8×A800 (≈ 8 × 80GB)					
CLIP	16.68 (26.33)	66.13 (75.58)	✗	✗	✗
OpenCLIP / Disco-CLIP	2.29 (23.56)	8.67 (26.89)	33.73 (62.37)	✗	✗
Inf-CL	0.21 (24.09)	0.41 (27.17)	0.81 (31.25)	1.63 (49.62)	✗
Inf-CL (w/ data offload)	0.21 (23.05)	0.41 (23.14)	0.81 (23.33)	1.63 (23.71)	6.53 (26.71)
32×A800 (≈ 32×80GB)					
CLIP	16.68 (26.33)	66.13 (75.58)	✗	✗	✗
OpenCLIP / Disco-CLIP	0.72 (19.28)	2.46 (19.88)	9.01 (21.42)	34.40 (50.24)	✗
Inf-CL	0.06 (19.32)	0.11 (19.92)	0.21 (21.09)	0.41 (23.49)	1.63 (52.43)

Table 1. **Training Memory Cost Across Different Hardware and Batch Sizes**. The baselines include the *Vanilla loss* (CLIP) and *Local loss* (OpenCLIP / Disco-CLIP). To minimize memory consumption of forward and backward, Gradient Cache is adopted, with an accumulation batch size of 32. Meanwhile, "data offload" strategy is proposed to reduce data memory usage (i.e., 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.

5. Experiments

5.1. Experimental Settings

Dataset and Data Processing. We assess the effectiveness of our **Inf-CL** on Laion400M dataset [28] where we used 280M (out of 400M) samples for training due to the unavailability of images in the remaining samples. Images undergo preprocessing using RandomResizedCrop with a crop ratio of $[0.75, 1.33]$ and a scale of $[0.08, 1.0]$.

Training Hyperparameters. A modified AdaFactor optimizer [29] is employed for training, following the settings of ViT-g [36]. The optimizer is configured with a learning rate of 1×10^{-3}, weight decay of 1×10^{-4}, and coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.95$ [38]. Training spans 8 epochs, using a cosine learning rate schedule with a linear warm-up during the first 0.5 epoch.

Implementation Details. For distributed training, we employ Data Parallelism [19] with Automatic Mixed Precision (float16) [22]. To reduce model cost, we adopt Gradient Cache [10] to decouple contrastive loss computation from the model's forward and backward. Consequently, the peak memory cost per iteration, M_{peak}, is calculated as:

$$M_{peak} \approx M_{data} + \max(M_{loss}, M_{backbone}), \tag{10}$$

where M_{data} is the memory for image and text data, M_{loss} is for loss computation, and $M_{backbone}$ is for the model's forward and backward operations.

Baselines. We compare our method against two baselines: the *vanilla loss* from CLIP and the *local loss* from OpenCLIP / DisCo-CLIP [1]. The *vanilla loss* computes a $b \times b$ similarity matrix by gathering both row and column features from all GPUs, while the *local loss* requires only column features to calculate a $b/n \times b$ similarity matrix, where b and n are the batch size and the number of GPUs.

[1]The *local loss* strategy in OpenCLIP is consistent with DisCo-CLIP.

5.2. Efficiency Results

To rigorously assess the memory efficiency of our method, we compare our approach with previous methods by evaluating "Memory Consumption","Maximum Batch Size" and "Speed" across various hardware settings. The valid memory cost is determined by peak memory (Equation 10), which is the maximum memory needed during an iteration.

Memory Consumption. To illustrate the memory efficiency of Inf-CL, we compared it to previous methods using the same batch size across different hardware configurations. Table 1 shows that Inf-CL requires significantly less memory than its predecessors, enabling our method to achieve batch sizes that previous methods could not handle (e.g., 1024k). Specifically, our method requires only 31.25 GB of peak memory (loss needs 0.81 GB) for a batch size of 128 and 8×GPUs, allowing us to complete 128k training using just 8×40G GPUs. In contrast, the previous SOTA consumes 62.37 GB and requires 8×80G GPUs to perform the same training. This demonstrates that our approach allows for more cost-effective training. Furthermore, we find that peak memory still increases rapidly with batch size due to growing data memory. For example, peak memory increases by 18.37 GB when the batch size grows from 128k to 256k on 8×GPUs. To address this issue, we integrate Inf-CL with "data offload" and mitigate this memory increase, enabling us to train a ViT-L/14 model with a batch size of 1024k on 8×A800.

Maximum Batch Size. We compare the maximum batch

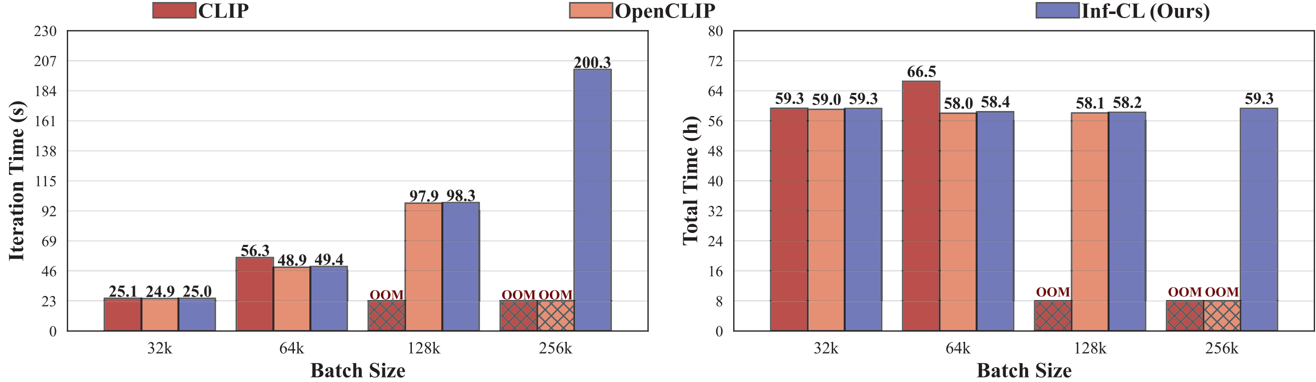

Figure 4. **Training Speed of ViT-L/14 CLIP on 8×A800 for Varying Batch Sizes.** The left figure shows the time per iteration step, while the right displays the time per epoch. Loss calculation contributes minimally to the total iteration time, making Inf-CL's iteration time comparable to previous methods.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

Method	GPUs	Max Batch Size	Memory (GB)
CLIP	8×A800	68k	74.39
OpenCLIP	8×A800	152k	59.95
Inf-CL	8×A800	448k	2.52
Inf-CL*	8×A800	4096k	26.12
CLIP	32×A800	68k	74.39
OpenCLIP	32×A800	352k	64.13
Inf-CL	32×A800	2048k	2.89
Inf-CL*	32×A800	12288k	19.59

Table 2. **Maximum batch size** for model training using different hardware and contrastive loss methods. The training settings are aligned with Table 1. * denotes adopting data offload.

size of Inf-CL with those of previous approaches under different training budgets (8×A800 or 32×A800) for ViT-L/14. As shown in Table 2. Inf-CL significantly outperforms previous SOTA methods, achieving improvements of 2.94× on 8×A800 (448k / 152k), which is further increased to 5.82× when using 32×A800 (2048k / 352k). Since Inf-CL has negligible memory requirements, peak memory is primarily driven by $M_{backbone} + M_{data}$. $M_{backbone}$ is constant, meaning the rapid growth in peak memory is mainly due to increased M_{data}. Since ViT-L/14 has a larger $M_{backbone}$, the remaining memory can accommodate only a small batch size for M_{data}. To address this issue, we introduce “data offload" to load only a small batch of data onto the GPU for each accumulation step, effectively stabilizing the data memory usage. As a result, we can scale the batch size to even 10M on 32×A800.

Training Speed. We compare the training speed of our Inf-CL with previous methods. As shown in Figure 4, using Inf-CL to train ViT-L/14 on 8×A800 has almost the same speed as previous methods. Even when increasing batch size beyond the limits of previous methods, Inf-CL maintains a linear increase in iteration time, with one epoch consistently taking about 59 hours. Inf-CL might be expected to exhibit much more slower performance because it breaks the loss calculation to small tiles and serially process these tiles. However, it achieves comparable or better speed to previous methods, as shown in Figure 4. This is due to two reasons (Detailed analysis in Appendix S1): (1) Inf-CL fuses the operations of similarity matrix calculation and softmax into a single communication for effectively reduces I/O time between HBM and DRAM; (2) Loss calculation represents only a minor fraction of the total iteration time. Combining training speed results with memory cost results demonstrates that Inf-CL has superior memory efficiency, while only introducing a little additional time cost.

5.3. Benchmark Results

In this section, we investigate whether training with Inf-CL is equivalent to vanilla CLIP and whether increasing batch size with Inf-CL enhances performance. We utilize the ViT-B/16 or ViT-B/32 with Bert-Base [9]. We follow the training strategy of LiT [37] to freeze the visual backbone and use the pre-trained weights instead.

Equivalence Verification. We evaluate CLIP models trained with different loss implementations, with the results presented in Table 3. As shown, under the same batch size, our Inf-CL performs similarly to previous methods, with performance differences falling within the error margin, confirming that our design incurs no precision loss in the loss calculations.

Batch Size Scaling. Furthermore, the results in Table 3 indicate that increasing the batch size within a certain range yields performance enhancements, thereby under-

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
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.** We use training strategies from 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 [8], ImageNet-v2 [26], ObjectNet [1] and ImageNet-OOD [15]. We also evaluate zero-shot image-text top-1 retrieval accuracy on MSCOCO [5].

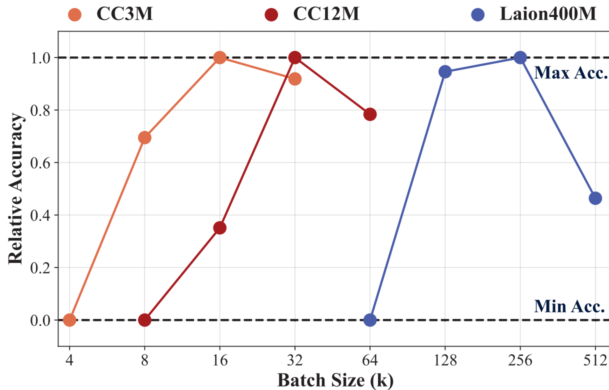

Figure 5. **Optimal Batch Size with Different Data Sizes** for ViT-B/32 LiT: As data size increases, the demand for a larger batch size also grows. The accuracy is normalized to $[0, 1]$, with magnitudes of 5.41 for CC3M, 1.85 for CC12M, and 2.78 for Laion400M.

Cross-GPU	In-GPU	Complexity	Memory
(Vanilla)		$\mathcal{O}(b^2)$	66.21
(OpenCLIP)		$\mathcal{O}(b^2/n)$	16.96
✔		$\mathcal{O}(b^2/n^2)$	4.81
✔	✔	$\mathcal{O}(b/n^2)$	0.81

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.

scoring the significance of our method for helping scale the batch size. However, under our experimental conditions, we currently observe that an excessively large batch size—previously unexamined in the literatures—results in suboptimal performance (i.e., 1024k). This may be attributed to factors such as unoptimized hyperparameters, inadequate training iterations, or constraints related to data size. As for hyperparameters and training iterations, corresponding analysis is put into Appendix S2). In Figure 5, We analyze the performance scaling trend with different batch sizes across various data sizes (e.g., CC3M, CC12M, and Laion400M). Specifically, performance on CC12M saturates at a batch size of 32k, whereas Laion400M achieves saturation at a batch size of 256k. The results indicate that larger datasets tend to have a correspondingly larger optimal batch size, demonstrating that our method shows greater advantages with larger-scale datasets.

5.4. Ablation Study

We ablate multi-level tiling in Table 4 and show that our designs incur no precision loss in loss calculations. This allows arbitrary combinations to achieve nearly the same zero-shot classification accuracy (about 74.8% on ImageNet for 64k batch size), while significantly reducing memory costs. According to the Equation 10, their M_{peak} is decided by $M_{backbone} + M_{data}$ rather than $M_{loss} + M_{data}$ as in prior methods. For complexity analysis, Cross-GPU tiling is $\mathcal{O}(b^2/n^2)$, resulting in a memory cost that is $1/n$ of OpenCLIP ($16.96/4.81 \approx 4$ in Table 4). Based on it, introducing In-GPU tiling can further reduce memory cost and make the growth of memory cost linear, i.e., $\mathcal{O}(b^2/n^2) \rightarrow \mathcal{O}(b/n^2)$.

6. Conclusion

This paper tackles the GPU memory bottleneck in scaling batch sizes for contrastive loss. To overcome the quadratic memory consumption resulting from the full instantiation of the similarity matrix, we proposed a tile-based computation strategy that partitions the calculation into smaller blocks, thus avoiding full matrix materialization. Furthermore, we introduced a multi-level tiling strategy that leverages ring-based communication and fused kernels to optimize synchronization and minimize I/O overhead. Our experiments demonstrated that our method scales contrastive loss batch sizes to unprecedented levels without compromising accuracy or training speed. We hope that the advancement of our method sheds light on further developments in areas such as self-supervised representation learning and text retrieval.

References

[1] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. *Advances in neural information processing systems*, 32, 2019. 8

[2] Changyou Chen, Jianyi Zhang, Yi Xu, Liqun Chen, Jiali Duan, Yiran Chen, Son Tran, Belinda Zeng, and Trishul Chilimbi. Why do we need large batchsizes in contrastive learning? A gradient-bias perspective. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. 2, 3

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. 1, 2

[4] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. *Advances in neural information processing systems*, 33:22243–22255, 2020. 2

[5] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015. 8

[6] Yihao Chen, Xianbiao Qi, Jianan Wang, and Lei Zhang. Disco-clip: A distributed contrastive loss for memory efficient clip training, 2023. 1, 3

[7] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. 2, 3

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 8

[9] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 7

[10] Luyu Gao, Yunyi Zhang, Jiawei Han, and Jamie Callan. Scaling deep contrastive learning batch size under memory limited setup, 2021. 1, 3, 6

[11] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings, 2022. 1

[12] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. Imagebind: One embedding space to bind them all. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15180–15190, 2023. 1

[13] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, pages 1735–1742, 2006. 1

[14] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2020. 1, 2

[15] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15262–15271, 2021. 8

[16] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, 2021. 1, 3

[17] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V. Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, pages 4904–4916. PMLR, 2021. 2

[18] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020. 2

[19] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020. 6

[20] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023. 2, 3

[21] Huaishao Luo, Lei Ji, Ming Zhong, Yang Chen, Wen Lei, Nan Duan, and Tianrui Li. Clip4clip: An empirical study of clip for end to end video clip retrieval and captioning. *Neurocomputing*, 508:293–304, 2022. 1

[22] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 6

[23] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 1

[24] Hieu Pham, Zihang Dai, Golnaz Ghiasi, Kenji Kawaguchi, Hanxiao Liu, Adams Wei Yu, Jiahui Yu, Yi-Ting Chen, Minh-Thang Luong, Yonghui Wu, et al. Combined scaling for open-vocabulary image classification. *arXiv preprint arXiv:2111.10050*, 1(2):4, 2021. 1, 3

[25] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. 1, 2, 3

[26] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International conference on machine learning*, pages 5389–5400. PMLR, 2019. 8

[27] Nikunj Saunshi, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar. A theoretical analysis of contrastive unsupervised representation learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 5628–5637. PMLR, 2019. 2, 3

[28] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs. *arXiv preprint arXiv:2111.02114*, 2021. 6

[29] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018. 6

[30] Nimit S. Sohoni, Christopher R. Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré. Low-memory neural network training: A technical report, 2022. 2

[31] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. 2

[32] Pavan Kumar Anasosalu Vasu, Hadi Pouransari, Fartash Faghri, Raviteja Vemulapalli, and Oncel Tuzel. Mobileclip: Fast image-text models through multi-modal reinforced training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15963–15974, 2024. 2

[33] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022. 1, 2

[34] Lilian Weng. Contrastive representation learning. *lilianweng.github.io*, 2021. 1

[35] An Yang, Junshu Pan, Junyang Lin, Rui Men, Yichang Zhang, Jingren Zhou, and Chang Zhou. Chinese clip: Contrastive vision-language pretraining in chinese. *arXiv preprint arXiv:2211.01335*, 2022. 2

[36] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12104–12113, 2022. 6

[37] Xiaohua Zhai, Xiao Wang, Basil Mustafa, Andreas Steiner, Daniel Keysers, Alexander Kolesnikov, and Lucas Beyer. Lit: Zero-shot transfer with locked-image text tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 18123–18133, 2022. 7

[38] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11975–11986, 2023. 2, 6

[39] Hang Zhang, Yeyun Gong, Yelong Shen, Jiancheng Lv, Nan Duan, and Weizhu Chen. Adversarial retriever-ranker for dense text retrieval. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 2

[40] Hang Zhang, Yeyun Gong, Xingwei He, Dayiheng Liu, Daya Guo, Jiancheng Lv, and Jian Guo. Noisy pair corrector for dense retrieval. *arXiv preprint arXiv:2311.03798*, 2023. 2