

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

Zesen Cheng<sup>1\*</sup> Hang Zhang<sup>1\*</sup> Kehan Li<sup>1\*</sup> Sicong Leng<sup>1</sup> Zhiqiang Hu<sup>1</sup>  
 Fei Wu<sup>2</sup> Deli Zhao<sup>1</sup> Xin Li<sup>1</sup> Lidong Bing<sup>3</sup>

<sup>1</sup> Alibaba Group, Hangzhou, China <sup>2</sup> Zhejiang University, Hangzhou, China

<sup>3</sup> Shanda AI Research Institute

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

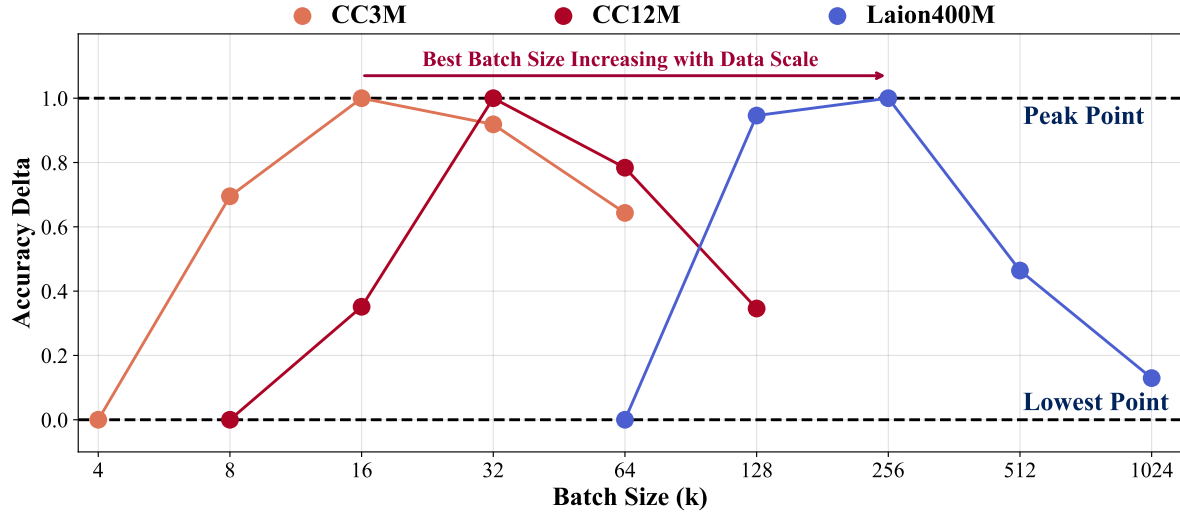


Figure S1. **Performance of ViT-B/32 across Varying Batch Sizes.** Except for batch size, other experiment settings are consistent. In Figure, the most suitable batch size is increasing with data scale.

In this document, we first analyze the training speed of our method (§Section S1). Then we describe the factors influencing performance when scaling batch size (§Section S2). Finally, a description of the multilevel and intra-gpu tile-wise global lse calculation backward process (§Section S6) are provided.

## S1. Analysis of Training Speed Efficiency

Although **Inf-CL** might be expected to exhibit slower performance because it breaks the loss calculation to small tiles and serially process these tiles, it achieves comparable or better speed to previous methods, as shown in Figure 4. This is primarily due to two factors: (1) Loss calculation represents only a minor fraction of the total iteration time, especially for large models, thereby exerting minimal impact on the overall iteration time. (2) While **Inf-CL** has similar computational complexity to standard contrastive loss, its tiling approach could introduce some speed overhead due to reduced parallelism. However, **Inf-CL** fuses

Batch Size	1×A800		8×A800		32×A800	
	Vanilla	Inf-CL	Vanilla	Inf-CL	Vanilla	Inf-CL
32k	28.73	28.17 ↓ 0.56	6.53	5.01 ↓ 1.52	3.10	1.49 ↓ 1.61
64k	-	58.50	15.85	10.56 ↓ 5.29	8.38	2.76 ↓ 5.62
128k	-	156.72	-	20.71	-	5.27
256k	-	332.13	-	45.98	-	11.27

Table S1. Training Speed (s) of ViT-B/32 CLIP on 1, 8, 32 × A800 for various batch sizes. “-” indicates out-of-memory error.

the operations of similarity matrix calculation and softmax, which in regular contrastive loss require two separate communications between SRAM and HBM. By merging these into a single communication, **Inf-CL** effectively reduces the I/O time, mitigating the cost of the serial tile computation.

Additionally, we have discovered that the implementation of Inf-CL for the vanilla version can be significantly accelerated. According to the results in Table S1, we observe

Batch Size	CC3M	CC12M	Laion400M
4k	47.60 ↑	-	-
8k	51.36 ↑	54.25 ↑	-
16k	<b>53.01</b>	54.90 ↑	-
32k	52.57 ↓	<b>56.10</b>	63.99 ↑
64k	51.08 ↓	55.70 ↓	66.62 ↑
128k	-	54.89 ↓	<b>66.77</b>
256k	-	-	65.28 ↓

Table S2. The absolute accuracy of ViT B/32 LiT on ImageNet val set using different batch sizes and different scale datasets.

that as the batch size increases, the acceleration of Inf-CL over the vanilla implementation of contrastive learning becomes more pronounced. Furthermore, the acceleration improves as the number of GPUs increases. This demonstrates that our method achieves high training speed efficiency.

## S2. Factors for scaling batch size

While a larger batch size is theoretically expected to improve performance [1], our experimental results deviate from this expectation. To better understand this discrepancy, we analyze the factors that impact performance when scaling up the batch size.

**Hyperparameters.** Although larger batch sizes provide more diverse negative samples for contrastive learning, potentially improving the embedding space, careful tuning of hyperparameters is necessary to ensure model convergence. Previous research indicates that when increasing batch size, the learning rate should be scaled proportionally to maintain a consistent parameter update norm throughout training [2]. Since a fixed learning rate is used across all experiments, this may have contributed to the reduced performance observed with larger batch sizes. Moreover, prior studies suggest that large batch sizes require longer training epochs to ensure sufficient parameter updates and avoid suboptimal convergence [3]. Overall, the performance gains from larger batch sizes are contingent on the careful tuning of multiple hyperparameters beyond just learning rate and epochs, highlighting the importance of comprehensive hyperparameter optimization to fully exploit the benefits of scaling.

**Data Scale.** Increasing batch size improves the precision of gradient estimation for the representation distribution defined by the dataset [1]. Larger datasets capture real-world distributions more accurately and thus employing a larger batch size allows contrastive loss to generate more precise gradients, enhancing the model’s ability to learn discriminative representations. As shown in Figure S1 and Table S2, our experiments on different data scales (e.g. CC3M, CC12M, and Laion400M) indicate that **the optimal**

Batch Size	4k	8k	16k	32k	64k
ViT-B/32	47.60 ↑	51.36 ↑	<b>53.01</b>	52.57 ↓	51.08 ↓
ViT-B/16	60.49 ↑	62.99 ↑	64.66 ↑	<b>64.89</b>	64.01 ↓

Table S3. The accuracy of ViT B/32 LiT and ViT B/16 LiT on ImageNet val set using different batch sizes.

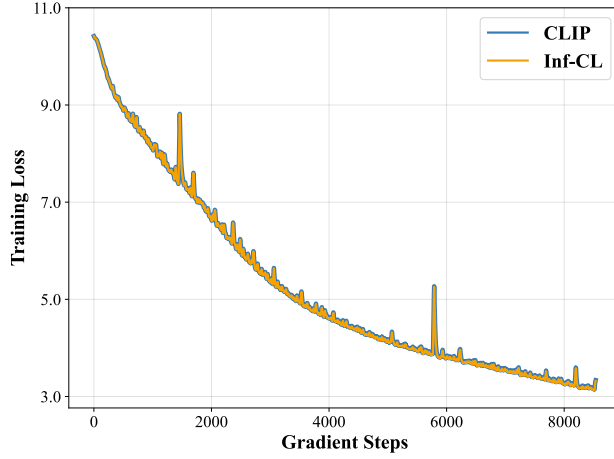
**batch size increases with dataset size.** Specifically, performance on CC12M saturates at a batch size of 32k, whereas Laion400M achieves saturation at a batch size of 256k.

**Model Size.** Our experimental results in Tab. S3 indicate that *the optimal batch size increases as the model size grows*. This could be attributed to the fact that larger models are more susceptible to noise in the gradients. Utilizing a larger batch size helps in stabilizing the gradients during the training of large models, thereby achieving better performance.

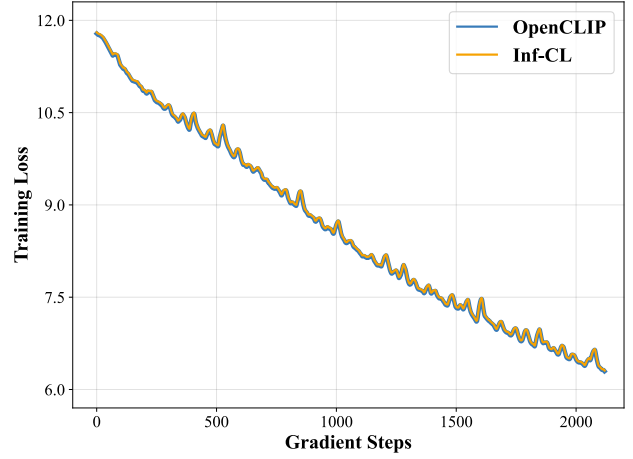
In summary, while scaling up batch sizes is critical for enhancing contrastive learning, our findings suggest that performance does not improve monotonically with increasing batch size. As seen in our previous experiments (Table 3), extremely large batch sizes (e.g. 1024k) can lead to a decrease in performance, indicating that factors such as hyperparameter tuning, dataset scale, and model size are among the many considerations that influence model effectiveness. This highlights the need for a balanced approach when increasing batch sizes, ensuring that optimal configurations are found to fully exploit the benefits of contrastive learning.

## S3. Discussion

Although our approach draws inspiration from Ring Attention and Flash Attention, implementing inf-cl is non-trivial. Inf-cl differs from these methods in terms of both implementation and contributions: 1). **Implementations:** The mentioned methods only consider single-level tiling calculations, while we develop a novel multi-level tiling strategy designed for contrastive learning to fit the corresponding large-scale distributed training system. Therefore, our approach can fully utilize parallelism across multiple GPUs and CUDA kernels to ensure efficiency while exploiting partial serial computation on a single GPU, eventually reducing memory costs to a negligible level without compromising processing speed. 2). **Contributions:** Beyond the new tiling implementation, another significant contribution is highlighting that tiling calculation can overcome the memory bottleneck that limits batch size scaling in contrastive learning, further expanding the application scope of the tile-based computing.



(a) CLIP vs Inf-CL (32k batch size)



(b) OpenCLIP vs Inf-CL (128k batch size)

Figure S2. **Training loss curve** on the Laion400M dataset. We set the same random seed to examine the loss curves when training with different loss functions (Vanilla loss, OpenCLIP loss, Inf-CL). As shown in the figure, the loss curves are nearly indistinguishable, showing that our Inf-CL does not impair convergence.

## S4. Derivation

In this section, we provide the detailed derivation of Eq. (4): Let  $\mathbf{l}_j^i$  denote the accumulated LSE value in  $j$ -th step and other symbols be the same as in the main text, the derivation for Eq. (4) is as follows:

$$\begin{aligned}
 \mathbf{l}_j^i &= \log \sum_{n=1}^j \sum_k e^{\mathbf{x}_{:,k}^{i,n}} = \log \left( \sum_{n=1}^{j-1} \sum_k e^{\mathbf{x}_{:,k}^{i,n}} + \sum_k e^{\mathbf{x}_{:,k}^{i,j}} \right) \\
 &= \log \left( \exp \left( \log \sum_{n=1}^{j-1} \sum_k e^{\mathbf{x}_{:,k}^{i,n}} \right) + \exp \left( \log \sum_k e^{\mathbf{x}_{:,k}^{i,j}} \right) \right) \\
 &= \log (e^{\mathbf{l}_{j-1}^i} + e^{\mathbf{l}_{j-1}^{i,j}}) \quad (\text{according to Eq. (5)}) \\
 &= \log e^{\mathbf{l}_{j-1}^i} + \log (1 + e^{\mathbf{l}_{j-1}^{i,j} - \mathbf{l}_{j-1}^i}) \\
 &= \mathbf{l}_{j-1}^i + \log (1 + e^{\mathbf{l}_{j-1}^{i,j} - \mathbf{l}_{j-1}^i})
 \end{aligned} \tag{S1}$$

## S5. Analysis of Loss Precision Error

To check whether our Inf-CL loss affects model convergence due to precision errors, we compare the loss curves of CLIP loss, OpenCLIP loss, and Inf-CL loss with the same random seed in Figure S2. As shown in the figure, whether with a small batch size (32k) or a large batch size (128k), our method shows almost no difference from the CLIP loss and OpenCLIP loss. This demonstrates that our Inf-CL loss significantly saves memory without affecting model convergence.

---

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

---

**Require:** Number of GPUs  $n$ , saved intermediate variables from the forward pass: in-memory visual features  $\mathbf{I}^i \in \mathbb{R}^{b_s \times c}$  and textual features  $\mathbf{T}^i \in \mathbb{R}^{b_s \times c}$  for each GPU, global LSE vectors  $\mathbf{l}^i \in \mathbb{R}^{b_s}$ .

- 1: Initialize vector:  $d\mathbf{I}^i = \mathbf{0} \in \mathbb{R}^{b_s \times c}$ ,  $d\mathbf{T}_{\text{cache}} = \mathbf{0} \in \mathbb{R}^{b_s \times c}$  on each GPU $_i$ .
- 2: **for**  $j = 1$  **to**  $n$  **do**
- 3:   **Asynchronously Text Feature Communication:**
- 4:   Each GPU sends in-memory textual feature to the next GPU and receive the textual feature from the previous GPU in the ring.
- 5:   **Backward Calculation:**
- 6:   Index of current text feature tile for each GPU:  $k = (i + j - 1) \bmod n$
- 7:   Call Algorithm 2 with  $(\mathbf{I}^i, \mathbf{T}^k, \mathbf{l}^i)$ , obtaining gradients  $d\mathbf{I}_{\text{temp}}^i$  and  $d\mathbf{T}_{\text{temp}}^k$ .
- 8:   Update gradients  $d\mathbf{I}^i += d\mathbf{I}_{\text{temp}}^i$ .
- 9:   Update gradients  $d\mathbf{T}_{\text{cache}} += d\mathbf{T}_{\text{temp}}^k$ .
- 10:   **Asynchronously Gradient Communication:**
- 11:   Each GPU sends in-memory  $d\mathbf{T}_{\text{cache}}$  to the next GPU in the ring.
- 12:   Each GPU receive the gradient feature from the previous GPU and write to  $d\mathbf{T}_{\text{cache}}$ .
- 13: **end for**
- 14:  $d\mathbf{T}^i = d\mathbf{T}_{\text{cache}}$  in each GPU.
- 15: Return the gradients  $d\mathbf{I}^i$ ,  $d\mathbf{T}^i$  for each GPU.

---

---

**Algorithm 2** Backward Process from of intra-GPU Tile-Wise LSE calculation

---

**Require:** Saved intermediate variables from the forward pass: visual features  $\tilde{\mathbf{I}} \in \mathbb{R}^{b \times c}$ , textual features  $\tilde{\mathbf{T}} \in \mathbb{R}^{b \times c}$ , the local LSE vector  $\tilde{\mathbf{l}} \in \mathbb{R}^b$ .

The row-wise and column-wise size of a tile:  $t_r$  and  $t_c$ ,

- 1: Divide  $\tilde{\mathbf{I}}$  into  $\tilde{\mathbf{I}}^i$ , where  $i = 1, 2, \dots, \tilde{n}_r$ .
  - 2: Divide  $\tilde{\mathbf{T}}$  into  $\tilde{\mathbf{T}}^j$ , where  $j = 1, 2, \dots, \tilde{n}_c$ .
  - 3: Divide  $\tilde{\mathbf{l}}$  into  $\tilde{\mathbf{l}}^i$ , where  $i = 1, 2, \dots, \tilde{n}_r$ .
  - 4: Initialize gradients vectors:  $d\tilde{\mathbf{I}} \in \mathbb{R}^{t_r \times c}$  and  $d\tilde{\mathbf{T}} \in \mathbb{R}^{t_c \times c}$ .
  - 5: **for** each  $\tilde{\mathbf{I}}^i$  **do**
  - 6:   Load  $\tilde{\mathbf{I}}^i$  and  $\tilde{\mathbf{l}}^i$  from HBM to on-chip SRAM.
  - 7:   Initialize  $d\tilde{\mathbf{I}}^i = \mathbf{0} \in \mathbb{R}^{t_r \times c}$ .
  - 8:   **for**  $j = 1$  **to**  $\lceil b/t_c \rceil$  **do**
  - 9:     Load  $\tilde{\mathbf{T}}^j$  from HBM to on-chip SRAM.
  - 10:    On chip, compute  $\tilde{\mathbf{X}}^{i,j} = \tilde{\mathbf{I}}^i \cdot \tilde{\mathbf{T}}^{j'}$   $\in \mathbb{R}^{t_r \times t_c}$ .
  - 11:    On chip, compute  $d\tilde{\mathbf{X}}^{i,j} = \exp(\tilde{\mathbf{X}}^{i,j} - \tilde{\mathbf{l}}^i) \in \mathbb{R}^{t_r \times t_c}$ .
  - 12:    Update gradients  $d\tilde{\mathbf{I}}^i += d\tilde{\mathbf{X}}^{i,j} \cdot \tilde{\mathbf{T}}^j$ .
  - 13:    Load  $d\tilde{\mathbf{T}}^j$  from HBM to on-chip SRAM.
  - 14:     $d\tilde{\mathbf{T}}^j += \tilde{\mathbf{I}}^i \cdot d\tilde{\mathbf{X}}^{i,j}$ .
  - 15:    Write updated  $d\tilde{\mathbf{T}}^j$  back to HBM.
  - 16:   **end for**
  - 17:   Write updated  $d\tilde{\mathbf{I}}^i$  back to HBM.
  - 18: **end for**
  - 19: **return**  $d\tilde{\mathbf{I}}$  (i.e.  $\frac{\partial \tilde{\mathbf{I}}}{\partial \tilde{\mathbf{I}}}$ ),  $d\tilde{\mathbf{T}}$  (i.e.  $\frac{\partial \tilde{\mathbf{I}}}{\partial \tilde{\mathbf{T}}}$ ).
- 

## S6. Backward Process

In this section, we describe the backward process of multi-level tiling in Alg. 1. Given the multi-level tiling, we employ an intra-GPU multi-kernel tiling strategy. We also provide additional details on the backward propagation process within the intra-GPU tiling strategy in Alg. 2.

## References

- [1] 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*. [2](#)
- [2] P Goyal. Accurate, large minibatch sg d: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. [2](#)
- [3] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017. [2](#)