1 Definitions: Contrastive Language–Image Pre-training (CLIP)

For the original paper, see Radford et al. (2021).

1.1 The Big Picture

1. **Goal**: Learn from (image, caption) pairs, with no fixed list of class labels, so that an image and a text describing the same thing land *close together* in one **shared latent space**, while mismatched images and texts land far apart. When this holds, we say the two modalities are **aligned**. A trained CLIP does not generate anything: it turns an image or a text into a vector and compares vectors.

2. **A training batch**: N image–text pairs $B = \{(I_1, T_1), \ldots, (I_N, T_N)\}$, where image i and text i belong together. These N pairs are **positives**. The other $N(N-1)$ combinations (I_i, T_j) with $j \neq i$ are treated as **negatives**.

1.2 Encoders and Translation Layers

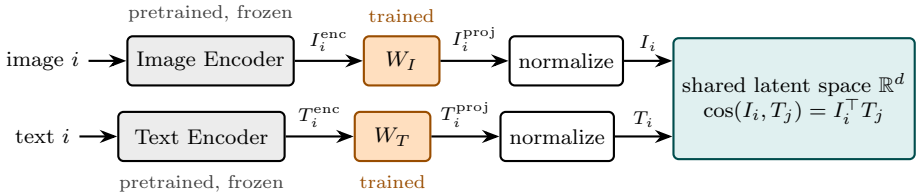
1. **Encoders**: The image encoder (a ResNet or Vision Transformer) extracts visual features, and the text encoder (a Transformer) summarizes the token sequence: $I_i^{\text{enc}} = \text{ImageEncoder}(\text{image } i)$ and $T_i^{\text{enc}} = \text{TextEncoder}(\text{text } i)$. Their outputs live in different spaces and may have different dimensions, so we cannot compare them directly.

2. **Translation layers**: Learned linear maps (*projection layers* in lecture) put both feature vectors into the same dimension d. This is the step that makes the comparison possible:

$$I_i^{\text{proj}} = W_I \, I_i^{\text{enc}} \in \mathbb{R}^d, \qquad T_i^{\text{proj}} = W_T \, T_i^{\text{enc}} \in \mathbb{R}^d. \tag{1.1}$$

3. **Normalization**: Scaling both embeddings to unit length removes the vector's length from the score. Otherwise the model could raise a score by making a vector longer instead of pointing it the right way. The dot product is then the cosine similarity:

$$I_i = \frac{I_i^{\text{proj}}}{\|I_i^{\text{proj}}\|_2}, \qquad T_i = \frac{T_i^{\text{proj}}}{\|T_i^{\text{proj}}\|_2}, \qquad I_i^\top T_j = \cos(I_i, T_j) \in [-1, 1]. \tag{1.2}$$



The encoders can be trained separately

Take a vision model and a language model that were each trained on their own, freeze them, and train only W_I and W_T. The encoders describe their own modality. The translation layers describe how the two relate, by putting them in the same latent space. (The original CLIP trains the encoders and translation layers together from scratch, with the same loss.)

1.3 The Contrastive Loss

1. **Score matrix**: Compute every image–text score at once, with $S_{ij} = I_i^\top T_j / \tau$ (the logit for pair (i, j)). The **diagonal** holds the positives, and everything off the diagonal is a negative. The temperature $\tau > 0$ controls the softmax: a smaller τ magnifies score differences. For example, cosines $[0.30, 0.25, 0.10]$ give probabilities $[0.36, 0.34, 0.30]$ with $\tau = 1$ but $[0.57, 0.35, 0.08]$ with $\tau = 0.1$. In CLIP, τ is learned, not hand-tuned.

	T_1	T_2	T_3	T_4
I_1	$I_1^\top T_1/\tau$	$I_1^\top T_2/\tau$	$I_1^\top T_3/\tau$	$I_1^\top T_4/\tau$
I_2	$I_2^\top T_1/\tau$	$I_2^\top T_2/\tau$	$I_2^\top T_3/\tau$	$I_2^\top T_4/\tau$
I_3	$I_3^\top T_1/\tau$	$I_3^\top T_2/\tau$	$I_3^\top T_3/\tau$	$I_3^\top T_4/\tau$
I_4	$I_4^\top T_1/\tau$	$I_4^\top T_2/\tau$	$I_4^\top T_3/\tau$	$I_4^\top T_4/\tau$

image-to-text, (1.3): softmax across row 2

text-to-image, (1.4): softmax down column 3

2. **Image-to-text (compare within a row)**: Fix image i and ask: *which of the N texts is my caption?* Each image solves an N-way classification problem whose correct class is $j = i$, so the true caption must outscore every other caption *for this image*:

$$p_{IT}(i \text{ matches } j \mid I_i, T_1, \ldots, T_N) = \frac{\exp(I_i^\top T_j/\tau)}{\sum_{k=1}^{N} \exp(I_i^\top T_k/\tau)}. \tag{1.3}$$

3. **Text-to-image (compare within a column)**: Fix text i and ask: *which of the N images do I describe?* The scores are the same but the competitors differ: the true image must outscore every other image *for this caption*. Below we drop the conditioning and write $p_{IT}(i \text{ matches } j)$ and $p_{TI}(i \text{ matches } j)$.

$$p_{TI}(i \text{ matches } j \mid T_i, I_1, \ldots, I_N) = \frac{\exp(I_j^\top T_i/\tau)}{\sum_{k=1}^{N} \exp(I_k^\top T_i/\tau)}. \tag{1.4}$$

4. **Why take the log?** We want the correct match to be likely for *every* pair at once, i.e., to maximize $\prod_i p_{IT}(i \text{ matches } i)$. The log keeps the same maximizer, but turns the product into a sum that does not underflow. It also undoes the exponential, and $-\log p$ punishes a confident mistake heavily while an already-correct pair costs almost nothing:

$$\ell_i^{IT} = -\log p_{IT}(i \text{ matches } i) = \underbrace{-\frac{I_i^\top T_i}{\tau}}_{\text{pull positive up}} + \underbrace{\log \sum_{j=1}^{N} \exp\Big(\frac{I_i^\top T_j}{\tau}\Big)}_{\text{push competitors down}}. \tag{1.5}$$

5. **The symmetric CLIP objective**: Average the cross-entropy over the batch in both directions, using the same score matrix:

$$\mathcal{L}_{IT} = -\frac{1}{N}\sum_i \log p_{IT}(i \text{ matches } i), \quad \mathcal{L}_{TI} = -\frac{1}{N}\sum_i \log p_{TI}(i \text{ matches } i), \quad \mathcal{L}_{\text{CLIP}} = \frac{\mathcal{L}_{IT} + \mathcal{L}_{TI}}{2}. \tag{1.6}$$

1.4 Using a Trained CLIP

1. **Off-the-shelf classifier (zero-shot)**: To classify a new image I_1 among C classes, write one sentence per class, encode and normalize the sentences to get $T_1, \ldots, T_C$, encode and normalize the image, and choose the sentence with the highest similarity. No new classifier is trained, and no labeled images are used. A softmax turns the scores into a distribution over that list:

$$p_{IT}(1 \text{ matches } c \mid I_1, T_1, \ldots, T_C) = \frac{\exp(I_1^\top T_c/\tau)}{\sum_{c'=1}^{C} \exp(I_1^\top T_{c'}/\tau)}, \qquad \hat{y} = \arg\max_c \; I_1^\top T_c. \tag{1.7}$$

For example, for a photo of a fox with $\tau = 0.1$:

Sentence	"this is a cat"	"this is a dog"	**"this is a fox"**
Cosine similarity	0.55	0.60	**0.80**
Probability	6.7%	11.1%	**82.1%**

2. **Prompt ensembling**: Different templates ("a photo of a {}", "a drawing of a {}", ...) give different class embeddings. Average the normalized template embeddings, then normalize again.

3. **Limitations**: CLIP aligns one global vector per image with one per sentence, so it can miss counting, negation, and object relationships.

2 What Does CLIP Learn?

1. A trained CLIP model is given one new image. What does it output for this image, and for an (image, text) pair? Can CLIP, on its own, write a caption for the image?

   For the image: a single unit-length vector $I \in \mathbb{R}^d$ (1.2), not a label or a caption. For a pair: a single number, the cosine similarity $I^\top T \in [-1, 1]$, which says how compatible they are. It cannot write a caption. CLIP has no decoder, so it can only *score* texts we give it. It can pick the best caption from a list, but generating text needs a language model attached to the image embedding.

2. **Alignment.**

   (a) Take an image encoder trained only for ImageNet classification and a text encoder trained only on text, and add randomly initialized translation layers W_I, W_T into the same $\mathbb{R}^d$. Before any further training, is the cosine $I^\top T$ between an image and a caption meaningful? What training makes it meaningful?

   No. Each encoder organizes its own space for its own task, so the dot product between them is essentially arbitrary. Training on *paired* data with the contrastive loss (1.6) forces the two to agree, so that an image and its caption point in the same direction. That training can update only W_I and W_T with the encoders frozen, or the encoders too, as in the original CLIP. Either way, the shared space is what gets aligned.

   (b) After training, we multiply every image embedding by the same rotation (orthogonal) matrix R, i.e., $I_i \mapsto RI_i$, and leave the text embeddings unchanged. Does image–text retrieval still work? What if we rotate *both* by the same R?

   Rotating only the images: in general, no, because $(RI_i)^\top T_j \neq I_i^\top T_j$. Rotating both: yes, because $(RI_i)^\top (RT_j) = I_i^\top R^\top R T_j = I_i^\top T_j$. The loss depends only on dot products *between* the two modalities, so CLIP learns their relative geometry, not a particular coordinate system.

3 Building the CLIP Loss by Hand

Consider a toy batch of $N = 3$ pairs with $d = 3$. After the encoders and translation layers (1.1), the (not yet normalized) embeddings are:

	$i = 1$	$i = 2$	$i = 3$
Image I_i^{proj}	(2, 0, 0)	(0, 3, 0)	(0, 0, 4)
Text T_i^{proj}	(4, 3, 0)	(6, 8, 0)	(0, 3, 4)

1. **Normalize and score.** Compute the unit-length embeddings using (1.2), then the 3×3 matrix of cosines $I_i^\top T_j$ (rows: images, columns: texts). Which entries are the positives? In each row, which caption is the hardest negative?

   $I_1 = (1, 0, 0)$, $I_2 = (0, 1, 0)$, $I_3 = (0, 0, 1)$; $T_1 = (0.8, 0.6, 0)$, $T_2 = (0.6, 0.8, 0)$, $T_3 = (0, 0.6, 0.8)$.
   $\left[I_i^\top T_j\right] = \begin{bmatrix} 0.8 & 0.6 & 0 \\ 0.6 & 0.8 & 0.6 \\ 0 & 0 & 0.8 \end{bmatrix}$. Positives: the diagonal, each the largest in its row and column.
   Hardest negatives: row 1, T_2 (0.6); row 2, a tie between T_1 and T_3 (0.6); row 3, T_1 and T_2 (0, both easy).

2. **Why normalize?** Compute the *raw* dot products $I_i^{\text{proj}\top} T_j^{\text{proj}}$ for all nine pairs. Which caption does image 1 prefer now? What goes wrong without normalization?

   $\left[I_i^{\text{proj}\top} T_j^{\text{proj}}\right] = \begin{bmatrix} 8 & 12 & 0 \\ 9 & 24 & 9 \\ 0 & 0 & 16 \end{bmatrix}$. Image 1 picks T_2 (12 vs. 8), only because T_2^{proj} is twice as long as T_1^{proj}. After normalization it picks its own caption (0.8 vs. 0.6), since the score then measures only direction.

3. **Temperature.** To keep the arithmetic simple, use $\tau = 1/(k \ln 2)$, so that every exponential is a power of 2: $\exp(I_i^\top T_j/\tau) = 2^{k\, I_i^\top T_j}$. Look only at image 1 (row 1) and fill in the table.

τ	**Logits** $I_1^\top T_j/\tau$	exp(logits)	$p_{IT}(1 \text{ matches } j)$ for $j = 1, 2, 3$
$\tau \to \infty$	$[0,\ 0,\ 0]$	$[1,\ 1,\ 1]$, sum 3	$\left[\frac{1}{3},\ \frac{1}{3},\ \frac{1}{3}\right]$
$\frac{1}{5 \ln 2} \approx 0.29$	$[4,\ 3,\ 0] \cdot \ln 2$	$[16,\ 8,\ 1]$, sum 25	$\left[\frac{16}{25},\ \frac{8}{25},\ \frac{1}{25}\right] = [0.64,\ 0.32,\ 0.04]$
$\frac{1}{10 \ln 2} \approx 0.14$	$[8,\ 6,\ 0] \cdot \ln 2$	$[256,\ 64,\ 1]$, sum 321	$\approx [0.80,\ 0.20,\ 0.003]$

   How does the distribution change as τ decreases? Does the predicted caption change?

   It becomes more skewed toward the top caption, because a smaller τ stretches the gaps between scores. The ratio $p_{IT}(1 \text{ matches } 1)/p_{IT}(1 \text{ matches } 2)$ goes $1, 2, 4$ and $p_{IT}(1 \text{ matches } 1)/p_{IT}(1 \text{ matches } 3)$ goes $1, 16, 256$. The prediction stays T_1, since dividing by a positive number keeps the order: τ changes the confidence, not the ranking.

4. **The loss in both directions.** Use $\tau = 1/(5 \ln 2)$, so that $\left[\exp(S_{ij})\right] = \begin{bmatrix} 16 & 8 & 1 \\ 8 & 16 & 8 \\ 1 & 1 & 16 \end{bmatrix}$. Compute $p_{IT}(i \text{ matches } i)$ for each row (1.3) and $p_{TI}(i \text{ matches } i)$ for each column (1.4) as fractions, and then $\mathcal{L}_{IT}$, $\mathcal{L}_{TI}$ and $\mathcal{L}_{\text{CLIP}}$ (1.6). You may use $\ln 2 \approx 0.693$, $\ln 3 \approx 1.099$ and $\ln 5 \approx 1.609$.

   Rows: $p_{IT}(1 \text{ matches } 1) = \frac{16}{25}$, $p_{IT}(2 \text{ matches } 2) = \frac{16}{32} = \frac{1}{2}$, $p_{IT}(3 \text{ matches } 3) = \frac{16}{18} = \frac{8}{9}$.
   Columns: $p_{TI}(1 \text{ matches } 1) = p_{TI}(2 \text{ matches } 2) = p_{TI}(3 \text{ matches } 3) = \frac{16}{25}$.
   Logs: $-\log \frac{16}{25} \approx 0.446$, $-\log \frac{1}{2} \approx 0.693$, $-\log \frac{8}{9} \approx 0.118$.
   $\mathcal{L}_{IT} \approx \frac{1}{3}(0.446 + 0.693 + 0.118) \approx 0.419$, $\mathcal{L}_{TI} \approx 0.446$, so $\mathcal{L}_{\text{CLIP}} \approx \frac{1}{2}(0.419 + 0.446) \approx 0.433$.

5. **Rows vs. columns.** Which of the six matching probabilities in part 4 is the lowest? Why is this pair worse in one direction than in the other, and what does this say about using both directions?

   $p_{IT}(2 \text{ matches } 2) = \frac{1}{2}$. In row 2, image 2 has *two* close competitors, T_1 and T_3 (0.6 each). In column 2, text 2 has only one, I_1 (0.6), so $p_{TI}(2 \text{ matches } 2) = \frac{16}{25}$. The same positive faces different competitors in its row and in its column. Using both directions means every pair must win as an image choosing a caption and as a caption choosing an image.

4 Zero-Shot Inference

1. A frozen CLIP model classifies an image using the prompts "a photo of a dog", "a photo of a cat" and "a photo of a car", with $\tau = 0.1$. The cosines between the image and the three prompts are $[0.32,\ 0.27,\ 0.05]$. Compute the class probabilities (1.7) and the predicted class. Now add a fourth prompt, "a photo of a wolf", with cosine 0.33. What are the new prediction and the new probability of "dog"? What does this show?

   Logits $[3.2,\ 2.7,\ 0.5]$ give probabilities $[0.597,\ 0.362,\ 0.040]$, so the prediction is **dog**.
   With wolf, logits $[3.2,\ 2.7,\ 0.5,\ 3.3]$ give dog 0.360, cat 0.218, car 0.024, wolf 0.398: prediction **wolf**.
   $p(\text{dog})$ drops from 0.60 to 0.36 although the image and the dog prompt did not change. The probabilities are relative to the softmax denominator, so the class list is part of the classifier.

2. Why is this called *zero-shot* classification? Why do we use the prompt "a photo of a {class}" instead of just the class name?

   No labeled images of the target classes are used and nothing is trained. The prediction $\hat{y} = \arg\max_c I_1^\top T_c$ is a linear classifier on I_1 whose weights are the prompt embeddings, which come from the text encoder, not from images. CLIP was trained on full captions, so a sentence looks like its training data, and a template can disambiguate a name, e.g., "crane" as a bird vs. a machine.

5 Who's on the Porch?

A camera on your porch runs CLIP on every frame to detect whether there is a **non-human animal** on the porch. Each sentence reads "this is a photo of ...". For one frame, with $\tau = 0.1$, the similarities are:

	T_1: an empty porch	T_2: a person on a porch	T_3: a dog	T_4: a cat	T_5: a raccoon
$I^\top T_j$	0.60	0.20	0.55	0.55	0.50

1. What does "choose the highest-similarity sentence" predict? Now sum the probabilities of T_3, T_4, T_5 to get $P(\text{animal})$. Why do the two approaches disagree? Would $P(\text{animal})$ change if we added more animal sentences?

   Highest is T_1, so it predicts no animal. Logits $(6, 2, 5.5, 5.5, 5)$ give probabilities $(0.385, 0.007, 0.233, 0.233, 0.142)$, so $P(\text{animal}) = 0.608$. The animal evidence is split across three sentences: each one loses to T_1, but the group wins. Yes, it would change, as the wolf example showed, so any threshold must be tuned for one fixed list.

2. A classmate proposes just two sentences: "a porch with an animal" and "a porch with no animal". Why is this risky?

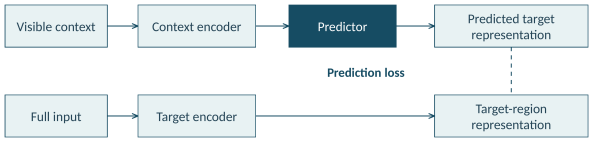
   CLIP can miss negation. The sentences share almost all their words, so their embeddings may be close. It is safer to describe what is actually in the scene.

3. At night the camera switches to infrared and accuracy drops. You freeze the encoders and train only the translation layers on captioned night frames. With $I_i^{\text{enc}} \in \mathbb{R}^{768}$, $T_i^{\text{enc}} \in \mathbb{R}^{512}$, and $d = 256$, how many entries do W_I and W_T have? The encoders have 120 million parameters. Why does this suit a small dataset?

> $256 \times 768 + 256 \times 512 = 327{,}680$ entries, about 0.27% of the model. There is very little to fit, so overfitting is less of a risk and few examples can be enough.

6 JEPA on the Porch

JEPA (Assran et al. 2023) starts from the same idea as a masked autoencoder: hide part of an image and try to fill it in. An autoencoder decodes back to pixels. JEPA has no decoder: it encodes the visible and hidden parts into a shared latent space and fills in the *representation* of the hidden part.



With s_x the context representation and s_y the target representation, $\hat{s}_y = \text{Pred}(s_x,\ \text{target location})$ and the loss $E = D(s_y, \hat{s}_y)$ compares features, not pixels. The target encoder follows a moving average of the context encoder.

1. Your porch video has blowing leaves, moving shadows, and rain. Why might filling in the latent space be better here than filling in the pixels?

> Exact pixels of leaves and rain are unpredictable, so a pixel loss spends effort on noise. In the latent space, the encoders can drop what cannot be predicted and keep high-level information, such as what is on the porch.

2. A JEPA predicts the features of the next frame, $\hat{s}_y = (0.60,\ 0.80)$, and $E = \|s_y - \hat{s}_y\|_2^2$ measures surprise. Compute E for each frame. Can this tell you that frame 3 contains a raccoon? How could you combine JEPA and CLIP in the detector?

Frame	1: leaves blowing	2: rain	3: something walks in
s_y	$(0.62,\ 0.78)$	$(0.57,\ 0.82)$	$(0.80,\ 0.60)$

> $E = 0.0008,\ 0.0013,\ 0.08$. No: a large E only says something unexpected happened, and JEPA's latent space has no connection to words. Run the CLIP classifier only on frames with a large E. JEPA finds when something happened, and CLIP says what it is.

3. You want to use your JEPA encoder with sentences instead of CLIP's image encoder. How would you connect them, and what would you train?

> Freeze the JEPA encoder and CLIP's text side. Add a translation layer W_J that maps JEPA features into CLIP's shared latent space, and train only W_J with the CLIP loss on (frame, caption) pairs.

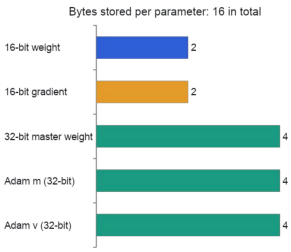
4. Suppose every encoder maps all inputs to one constant vector. What is E? What is $\mathcal{L}_{\text{CLIP}}$ for a batch of N pairs? What does this say about the role of negatives?

> JEPA: $E = 0$, a perfect loss with a useless representation (collapse). CLIP: every softmax is uniform, so $\mathcal{L}_{\text{CLIP}} = \log N$, a bad loss. Negatives make collapse expensive in CLIP. JEPA has none, so it needs other ways to keep information in the features, such as its moving-average target encoder.

7 Definitions: Distributed Training

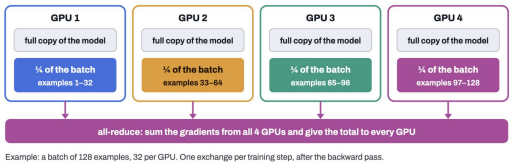
GPU size vs. Adam size: Adam keeps two running averages, m and v, for *every* parameter, plus a 32-bit master copy of each weight. Training with N parameters takes 16 bytes each, before activations:

$$16N = \underbrace{2N}_{\text{weights}} + \underbrace{2N}_{\text{gradients}} + \underbrace{12N}_{\text{Adam}}$$



Llama 3 8B needs 16 GB to run, so it fits on an 80 GB H100. Training it needs 128 GB, so it does not.

Data parallelism: Copy the whole model onto every GPU and split each batch. Each GPU computes its own gradients, then an **all-reduce** ($2N$ bytes, once per step) gives every GPU the same sum, so the copies stay identical. Each GPU still holds $16N$ bytes: more GPUs add speed but no memory.

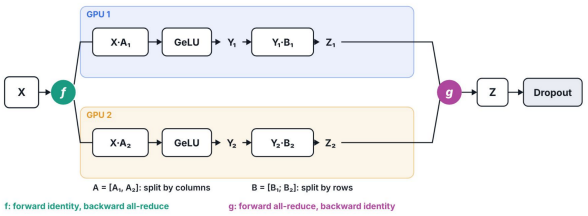


Tensor parallelism: Split each matrix multiplication *inside* a layer across p GPUs. Each GPU stores only its slice of the weights, with the gradients and Adam state for that slice, so memory per GPU is $16N/p$. For $Y = XW$, we can cut W (called A in the figures) by columns, which needs a sum in the backward pass, or by rows, which needs a sum in the forward pass.



	Column-parallel ($W_i \in \mathbb{R}^{D\times H/p}$)	**Row-parallel** ($W_i \in \mathbb{R}^{D/p\times H}$)
Forward	$Y = [XW_1, \ldots, XW_p]$	$Y = \sum_i X_i W_i$ **(all-reduce)**
Backward	$\partial\ell/\partial X = \sum_i G_i W_i^\top$ **(all-reduce)**	$\partial\ell/\partial X_i = G\,W_i^\top$

In a block: Split A by columns, because GeLU is not linear and cannot be applied to partial sums. Then split B by rows, so $Z = Y_1B_1 + Y_2B_2$. Operator f sums backward and g sums forward, so each block needs one sum each way.



Cost: Attention and MLP are two blocks, so each layer needs **4 all-reduces**, each moving (tokens × hidden size × 2) bytes no matter how large p is. So tensor parallelism stays on fast NVLink, inside one server.

Others: **Pipeline**: each GPU owns consecutive layers; idle share is $(p-1)/(m+p-1)$ for m micro-batches. **Optimizer** (ZeRO): shard the duplicated Adam state, then gradients, then weights. **Token** (TeraPipe): the next stage starts on early tokens while later ones finish. **Expert** (MoE): a router sends each token to a few experts through an all-to-all exchange.

8 Training PorchLM

You now want a language model that writes a nightly log of what the camera saw. Use these numbers:

PorchLM parameters	$N = 16 \times 10^9$	H100 memory	80 GB
Transformer layers	$L = 48$	Achieved compute	400 TFLOP/s per GPU
Hidden size	5,000	NVLink	450 GB/s
Tokens per micro-batch	9,000	Network	50 GB/s per GPU

Assume 16 bytes per parameter, 8 GPUs per server, and 1 GB $= 10^9$ bytes, and ignore activation memory. Training costs $6N$ FLOPs per token. Moving M bytes takes M/B, and an all-reduce takes about $2M/B$ (one pass to sum, one to send the total back), matching lecture's 2.5 GB in about 10 ms over NVLink.

1. **GPU size vs. Adam size**: What is the total training memory, and how much of it is optimizer state? Can one H100 run PorchLM for inference? Can it train PorchLM?

   Total: $16N = 256$ GB, of which $12N = 192$ GB (75%) is optimizer state. Inference needs $2N = 32$ GB, which fits on one H100. Training does not, since $256 > 80$ GB.

2. **Data parallelism**: What is the memory per GPU on 8 H100s? On 64? What is the largest model data parallelism can train on an H100?

   256 GB per GPU in both cases, because every GPU holds the full copy. The largest model has $80 \times 10^9/16 = 5 \times 10^9$ parameters.

3. You shrink to $N = 4 \times 10^9$. Eight H100s each process 9,000 tokens per step. Find the memory per GPU, the compute time, and the gradient all-reduce time over NVLink. What is the speedup over one GPU processing all 72,000 tokens?

   Memory: 64 GB, which fits. Compute: $6 \times (4 \times 10^9) \times 9{,}000/(400 \times 10^{12}) = 540$ ms.
   All-reduce: 2×8 GB$/(450$ GB$/$s$) \approx 35.6$ ms. Step: 575.6 ms.
   One GPU: $8 \times 540 = 4{,}320$ ms, so the speedup is $4{,}320/575.6 \approx 7.5\times$.

4. **Tensor parallelism by hand**: Let $p = 2$, $X = \left[\begin{smallmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 3 & 1 \end{smallmatrix}\right]$, and $W = \left[\begin{smallmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 2 & 1 \end{smallmatrix}\right]^\top$. Compute what each GPU produces for $Y = XW$ under column-parallel, and under row-parallel with the first two rows of W on GPU 1. If the next step squares each entry, compare entry $(1, 1)$ with squaring before and after the row-parallel sum. Which split must come first in an MLP?

   Column: $Y_1 = (6, 3)^\top$ and $Y_2 = (3, 8)^\top$, placed side by side with no sum.
   Row: $X_1 W_1 = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}$ and $X_2 W_2 = \begin{bmatrix} 1 & 1 \\ 1 & 7 \end{bmatrix}$, which sum to $Y = \begin{bmatrix} 6 & 3 \\ 3 & 8 \end{bmatrix}$.
   Before: $5^2 + 1^2 = 26$. After: $6^2 = 36$. The activation is not linear, so column-parallel must come first: each GPU then holds complete entries of Y.

5. Back to PorchLM. What is the smallest p that fits on H100s? With $p = 8$, what is the memory per GPU? Why does the Adam state shrink here when it did not under data parallelism?

   $256/80 = 3.2$, so $p = 4$. With $p = 8$: 32 GB per GPU, leaving 48 GB for activations. Adam keeps its numbers per parameter, and each GPU now owns only $1/p$ of the parameters.

6. Train PorchLM with $p = 8$ on one server. How many bytes does one all-reduce move, and how long does it take? What is the total communication time per micro-batch? What are the compute time, the step time, and the share of the step spent communicating?

   $M = 9{,}000 \times 5{,}000 \times 2 = 90$ MB, so one sum takes 2×90 MB$/(450$ GB$/$s$) = 0.4$ ms.
   Sums: $4 \times 48 = 192$, so communication takes $192 \times 0.4 = 76.8$ ms.
   Compute: $6 \times (16 \times 10^9) \times 9{,}000/(8 \times 400 \times 10^{12}) = 270$ ms.
   Step: $270 + 76.8 = 346.8$ ms, of which $76.8/346.8 \approx 22\%$ is communication.

7. A teammate wants $p = 16$ across two servers: "Twice the GPUs means half the compute." Assume every sum now runs at network speed. Find the new step time. Was your teammate right?

   Compute: 135 ms. One sum: 2×90 MB$/(50$ GB$/$s$) = 3.6$ ms, so communication takes $192 \times 3.6 = 691.2$ ms.
   Step: 826.2 ms, about $2.4\times$ **slower** on twice the hardware. Tensor parallelism should stay inside one server.

8. **Putting it together**: Use tensor parallelism with $p = 8$ inside each of 4 servers, and data parallelism across them. Each server holds one full copy of the model and processes its own 9,000 tokens. At the end of the step, each GPU all-reduces its gradient slice over the network with its match in the other servers. Find the memory per GPU, the size and time of that all-reduce, and the new step time. What is the speedup in tokens per second over one server?

   Memory is still 32 GB per GPU. Slice: $2N/8 = 4$ GB, so the all-reduce takes 2×4 GB$/(50$ GB$/$s$) = 160$ ms.
   Step: $346.8 + 160 = 506.8$ ms.
   One server: $9{,}000/0.3468 \approx 25{,}950$ tokens/s. Four servers: $36{,}000/0.5068 \approx 71{,}030$ tokens/s.
   Speedup: about $2.7\times$ on $4\times$ the hardware. The rest goes to the exchange over the slower network.