Lecture 17: Multimodal Models

Language Modeling from Scratch · lecture notes · rebuilt for NVIDIA Blackwell, late 2026

Section 1: The Multimodal Interface and Three Architectural Paradigms

A text-only language model receives a sequence of text tokens, runs them through a Transformer backbone, and emits a sequence of text tokens. Frontier models, however, are expected to operate across all human communication channels simultaneously rather than text alone. The ultimate goal is an **omni model** that accepts any combination of text, images, video, and audio as input and produces any combination of those modalities as output.

> **DEFINITION**
> **Omni model:** A single multimodal system built around a shared Transformer backbone that accepts any combination of modalities (text, images, video, audio) as input for understanding and produces any combination of modalities as output for generation (any-to-any reasoning).

Picture the structural shift from a text-only language model to an omni model as two stacked system diagrams. In the top row, representing the text-only language models studied so far, a single input box labeled **Text** feeds via one horizontal arrow into a **Language Model** box, which feeds via one horizontal arrow out to a single **Text** output box. In the bottom row, representing the **Omni Model**, four stacked input boxes on the left—**Text**, **Images**, **Video**, and **Audio**—all send converging arrows into a larger central **Omni Model** block housing a **Shared Transformer Backbone (any-to-any reasoning)**. The left-hand converging arrows represent **Understanding** (encoding incoming modalities into the backbone). From the right edge of the shared Transformer, four diverging arrows fan out into four stacked output boxes—**Text**, **Images**, **Video**, and **Audio**—representing **Generation**.

Because Transformers remain the core computational engine of an omni model, every modality must pass through one strict architectural bottleneck: Transformers operate exclusively on sequences of **tokens**, where each token is either a discrete vocabulary ID mapped to an embedding vector or a continuous vector injected directly into the residual stream, carrying a roughly semantic unit of information.

> **DEFINITION**
> **Visual token:** A discrete codebook index or continuous embedding vector representing a spatial region (such as a patch or merged group of patches) of an image or video frame inside a Transformer's input sequence.

For text, recall that Byte Pair Encoding (BPE) tokenization already converts raw byte streams into compact, semantically meaningful chunks. Non-text modalities resist this conversion for three structural reasons:

- **Continuous, high-dimensional, and redundant raw signals:** Pixels, video frames, and audio waveforms are dense arrays of continuous measurements in which adjacent samples are heavily correlated.
- **Lower information density than text:** A brief sentence takes a handful of BPE tokens, whereas a single image or short video clip expands into hundreds or thousands of tokens to preserve comparable content.
- **Variable geometry and duration:** Images arrive at arbitrary resolutions and aspect ratios, and video and audio span arbitrary durations; none of these map cleanly onto a fixed-length one-dimensional sequence.

COMMON MISCONCEPTION

You might think that a token entering a Transformer must always be a discrete integer index drawn from a finite vocabulary, just like a BPE text token. That breaks down in multimodal architectures because a Transformer's first layer simply consumes a sequence of d-dimensional real vectors; an external encoder can produce continuous vectors directly and concatenate them into the same sequence as text-token embeddings, without ever discretizing them into integer IDs.

WORKED EXAMPLE

Consider comparing the sequence length and self-attention cost of representing a single 336×336 RGB photograph versus a short caption describing it (`"A red bicycle parked beside a brick wall."`, roughly 10 BPE tokens).

1. **Raw pixel representation:** The image contains $336 \times 336 = 112{,}896$ pixels (and $112{,}896 \times 3 = 338{,}688$ scalar color values). Recall that dense self-attention computes pairwise dot products across all L sequence positions, scaling as $O(L^2)$. Treating each pixel as one token yields $L^2 = (112{,}896)^2 \approx 1.27 \times 10^{10}$ pairwise attention scores for a single image, which is computationally intractable.
2. **Spatial patch representation:** Grouping pixels into non-overlapping 14×14 patches produces $(336/14) \times (336/14) = 24 \times 24 = 576$ visual tokens. The pairwise attention matrix shrinks to $576^2 = 331{,}776$ entries—a reduction of $(112{,}896/576)^2 = 196^2 = 38{,}416\times$.
3. **Information density contrast:** Even after this $196\times$ spatial compression, one image still occupies 576 tokens—nearly $60\times$ more tokens than the 10-token caption summarizing it.

This single calculation explains why multimodal systems must compress raw signals before attention and still contend with dramatically longer sequences than text-only models.

Two guiding questions organize the design space of multimodal systems:

1. How do we **encode** non-text inputs into a Transformer so the model can understand them?
2. How do we **output** non-text modalities from a Transformer so the model can generate them?

Across modern research and deployed systems, answers to these two questions organize into **three architectural paradigms**, which can be visualized as three side-by-side vertical pipelines:

1. **Modular VLM (e.g., LLaVA, Qwen-VL series):** From top to bottom, a **Continuous Vision Encoder** extracts dense feature vectors from an image, passes them through a **Projector / Adapter** into an **LLM Backbone**, and produces **Text Output Only**. This design excels at visual comprehension but cannot natively generate images.
2. **Early-Fusion Discrete (e.g., Chameleon):** From top to bottom, a **VQ-VAE + BPE Tokenizer** converts both images and text into discrete integer IDs from a shared vocabulary, feeding a **Unified Autoregressive Transformer** that outputs **Text + Image Token IDs**; emitted image IDs pass through a **VQ-VAE Decoder** to reconstruct **Pixels**. A single next-token objective handles both reading and writing, at the cost of discretization error and training instability.
3. **Hybrid Continuous + Diffusion (current omni direction):** From top to bottom, **Continuous Encoders** feed rich semantic features into a **Multimodal Transformer** (the shared reasoning core), whose output splits into two parallel branches: a **Text LM Head** on the left for language generation and a **Diffusion Decoder** on the right for high-fidelity visual synthesis.

These three pipelines highlight the central tension running through the entire lecture: continuous features excel at comprehension, whereas generation requires reconstructing fine-grained pixel details.

KEY IDEA

Transformers require every modality to enter as a sequence of tokens (continuous vectors or discrete IDs), forcing a fundamental trade-off between continuous representations that maximize comprehension and discrete or generative decoders that reconstruct fine-grained details.

To understand how these three paradigms actually work, we must first solve the encoding problem: how do we build a vision encoder that transforms raw pixels into semantically meaningful representations aligned with language?

✎ Check Your Understanding

1. **Fill in the blank:** In the Modular VLM paradigm (such as LLaVA or the Qwen-VL series), a continuous vision encoder feeds features through a _____ into an LLM backbone, which produces _____ output only.

   Answer: **Projector / Adapter**; **text**. Modular VLMs align continuous visual embeddings to the LLM's input space while keeping a text-only output vocabulary.

2. **Yes or No:** Does converting an image into non-overlapping 14×14 patches make its information density per token equal to that of BPE-tokenized text? Justify your answer in one to two sentences.

   Answer: **No.** Even after grouping 14×14 pixels into each patch, a single 336×336 image requires 576 visual tokens, whereas a text description of the same scene takes only a few dozen BPE tokens, so images remain far less information-dense per token than text.

Section 2: Contrastive Vision Encoders — ViT Patching and CLIP

Classical computer vision trained image encoders on manually annotated label sets—most famously ImageNet, containing 1.2M labeled images across a fixed catalog of object classes. That supervision paradigm breaks down for general-purpose multimodal intelligence: human labeling cannot scale to every concept in the world, and predicting a closed list of fixed class indices prevents a vision encoder from capturing open-ended relationships, attributes, actions, or novel objects.

CLIP (Contrastive Language-Image Pretraining, 2021—the foundational baseline for modern multimodal encoders) asked a different question: can we learn general visual semantics from abundant, noisy `(image, caption)` pairs scraped from the web instead of curated class labels?

> **DEFINITION**
> **Contrastive Language-Image Pretraining (CLIP):** A dual-encoder pretraining method that maps images and natural-language captions into a shared normalized embedding space by training matching pairs to have high cosine similarity and mismatched in-batch pairs to have low similarity.

To construct its training data, CLIP searched 500K text queries and collected roughly 20K `(image, text)` pairs per query, assembling 400M image-text pairs (a private dataset that was never released). The open community later reproduced CLIP in **OpenCLIP** using **LAION-5B**, a web-scale dataset that itself relied on CLIP similarity scores to filter noisy web scrapes. Supervising vision with natural language immediately replaces a closed label table with open-vocabulary semantics.

Before aligning images with text, the image encoder itself must convert a two-dimensional pixel grid into a feature vector. CLIP's best-performing vision backbone—outperforming convolutional ResNet-50 baselines—was the **Vision Transformer** (`ViT-L/14@336px`, a Large ViT using 14×14 patches over 3 RGB channels at 336×336 resolution).

> **DEFINITION**
> **Vision Transformer (ViT) patch embedding:** The front-end projection of a Vision Transformer that splits a preprocessed RGB image into non-overlapping square pixel patches, flattens each patch into a vector, applies a learned linear projection, and adds 2D positional embeddings to form a sequence of visual tokens.

DEFINITION

Attention pooling: A single query-key-value attention layer placed at the end of a Vision Transformer whose query is the global average of all patch activations, aggregating the entire spatial token grid into one global image vector.

Trace the five-stage pipeline of `ViT-L/14@336px` from left to right:

1. **Raw Image (arbitrary $W \times H$):** The input arrives at whatever height and width the photograph happens to have.
2. **Resize + Center Crop (336×336 px grid):** The image undergoes bicubic resizing so its shorter side equals 336 pixels, followed by a center crop to a strict 336×336 square—which physically cuts off content along the borders of non-square images. The cropped square is partitioned into a regular grid of 14×14 pixel squares across 3 RGB channels (one highlighted corner cell illustrates a single $14 \times 14 \times 3$ patch).
3. **Linear Patch Embedding (+ 2D pos. emb.):** Each $14 \times 14 \times 3$ patch is flattened and multiplied by a shared linear projection matrix, then summed with a 2D positional embedding so the Transformer knows where each patch sat in the grid. This produces $24 \times 24 = 576$ visual tokens.
4. **ViT-L Transformer (bidirectional attention):** Unlike a causal decoder LM, every visual token attends to every other visual token in both directions across the ViT layers, letting patches build global visual context.
5. **Attention Pooling ($Q =$ global avg):** A final single QKV attention layer uses the average of all 576 patch activations as its query to attend over the patch keys and values, collapsing the spatial grid into **1 image vector**. (We will revisit this exact step in Section 4: later VLMs drop this final attention pooling layer and feed the full spatial grid of ViT patch embeddings directly into the LLM.)

WORKED EXAMPLE

Trace the preprocessing and patch geometry of `ViT-L/14@336px` on a landscape photograph of resolution 1008×672 pixels (3 RGB channels):

1. **Bicubic resize:** The shorter side is 672 px. Scaling so the shorter side becomes 336 px multiplies both dimensions by $336/672 = 0.5$, yielding a 504×336 image.
2. **Center crop:** Cropping the central 336×336 square trims $(504 - 336)/2 = 84$ px from the left border and 84 px from the right border. Exactly $168/504 = 33.3\%$ of the original photograph's horizontal field of view is discarded before the neural network ever sees a pixel.
3. **Patch slicing and flattening:** Dividing 336 by the patch size 14 gives 24 patches horizontally and 24 patches vertically, for $24 \times 24 = 576$ tokens. Each patch contains $14 \times 14 \times 3 = 588$ scalar pixel intensities.
4. **Linear projection:** A single projection matrix of shape $d_{\text{ViT}} \times 588$ maps each 588-element flattened patch into a d_{ViT}-dimensional token vector, to which its 2D positional embedding is added before entering the bidirectional Transformer layers.

Notice the built-in pitfall exposed by step 2: if a document, chart legend, or street sign sits near the left or right edge of a wide image, center-cropping deletes it outright.

With a ViT turning each image into one vector, CLIP pairs it with a **Text Encoder**—a GPT-2 Transformer (63M parameters, 12 layers) that encodes `[BOS] ... [EOS]` and returns the top-layer `[EOS]` activation as the caption's summary vector.

To train both encoders jointly, CLIP samples a large batch of N aligned `(image, text)` pairs (for example, $N = 32{,}768$) and forms an $N \times N$ **pairwise similarity matrix**. Visualize this grid with the N image embeddings $I_1, I_2, \ldots, I_N$ running down the vertical axis (rows, produced by the Image Encoder) and the N text embeddings $T_1, T_2, \ldots, T_N$ running across the horizontal axis (columns, produced by the Text Encoder). The N highlighted cells along the main diagonal—$I_1 \cdot T_1, I_2 \cdot T_2, \ldots, I_N \cdot T_N$—correspond to the true matching `(image, caption)` pairs. All $N^2 - N$ shaded off-diagonal cells are mismatched negative pairs. Training pushes the N diagonal entries up while pushing down all $N^2 - N$ off-diagonal entries through a **symmetric objective**:

- **Row-wise (image $\rightarrow$ text):** Each image must select its true caption out of the N captions in the batch (1 positive versus $N-1$ distractors).
- **Column-wise (text $\rightarrow$ image):** Each caption must select its true image out of the N images in the batch (1 positive versus $N-1$ distractors).

Formally, project the raw encoder outputs into a shared d-dimensional space and L_2-normalize them to unit vectors $\hat{v}_i, \hat{t}_j \in \mathbb{R}^d$ (so $\|\hat{v}_i\|_2 = \|\hat{t}_j\|_2 = 1$). Introduce a learnable positive inverse-temperature scale $\tau = e^t$ (parameterized via the unconstrained scalar t so $\tau > 0$ always), and define the similarity logits:

$$S_{ij} = \tau\, \hat{v}_i^\top \hat{t}_j$$

The CLIP symmetric cross-entropy loss is:

$$\mathcal{L}_{\text{CLIP}} = -\frac{1}{2N} \sum_{i=1}^{N} \left[\log \frac{e^{S_{ii}}}{\sum_{j=1}^{N} e^{S_{ij}}} + \log \frac{e^{S_{ii}}}{\sum_{j=1}^{N} e^{S_{ji}}} \right]$$

Decomposing this equation hierarchically from the innermost operation outward:

1. **Cosine similarity ($\hat{v}_i^\top \hat{t}_j$):** Because both vectors have unit L_2 norm, their inner product equals the cosine of the angle between image i and caption j, bounded in $[-1, 1]$.
2. **Temperature-scaled logit ($S_{ij} = \tau\, \hat{v}_i^\top \hat{t}_j$):** Without τ, logits bounded in $[-1, 1]$ can never produce sharp softmax probabilities near 1; multiplying by the learned scale $\tau = e^t$ lets the model sharpen the probability distribution as alignment improves.
3. **Row-wise softmax probability ($\frac{e^{S_{ii}}}{\sum_{j=1}^{N} e^{S_{ij}}}$):** Fixing image i (row i), the denominator sums $e^{S_{ij}}$ across all N captions in the batch. The ratio is the model's predicted probability that caption i is the true match for image i. Taking $-\log(\cdot)$ is the standard cross-entropy loss for image-to-text retrieval.
4. **Column-wise softmax probability ($\frac{e^{S_{ii}}}{\sum_{j=1}^{N} e^{S_{ji}}}$):** Fixing caption i (column i), the denominator sums $e^{S_{ji}}$ across all N images in the batch. Taking $-\log(\cdot)$ is the cross-entropy loss for text-to-image retrieval.
5. **Symmetric batch average ($-\frac{1}{2N} \sum_{i=1}^{N} [\dots]$):** Summing both directional losses over all N pairs and dividing by $2N$ averages the image-to-text and text-to-image cross-entropies equally.

As a single conceptual unit, $\mathcal{L}_{\text{CLIP}}$ treats contrastive alignment as $2N$ simultaneous N-way classification problems—forcing every image to rank its own caption above all $N-1$ other captions in the batch, and every caption to rank its own image above all $N-1$ other images.

In PyTorch, this entire objective executes in seven lines:

```
I_e = F.normalize(image_encoder(I) @ W_i, dim=-1)  # [N, d]
T_e = F.normalize(text_encoder(T)  @ W_t, dim=-1)  # [N, d]

logits = (I_e @ T_e.T) * torch.exp(t)              # [N, N]
labels = torch.arange(N)
loss_i = F.cross_entropy(logits,   labels)         # image -> text
loss_t = F.cross_entropy(logits.T, labels)         # text  -> image
loss   = 0.5 * (loss_i + loss_t)
```

Lines 1–2 encode the batch, apply learned linear projections `W_i` and `W_t`, and L_2-normalize along the embedding dimension. Line 4 computes the full $N \times N$ matrix of scaled cosine similarities `S`. Line 5 sets the ground-truth class for row i and column i to index i (the diagonal). Lines 6–8 compute row-wise and column-wise cross-entropy (transposing `logits` swaps columns into rows) and average them.

WORKED EXAMPLE

Evaluate $\mathcal{L}_{\text{CLIP}}$ step by step for a tiny batch of $N = 2$ pairs in $d = 2$ with learnable scale $\tau = e^t = 2.0$. Suppose the L_2-normalized embeddings are:

$$\hat{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \hat{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \hat{t}_1 = \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}, \quad \hat{t}_2 = \begin{bmatrix} 0.28 \\ 0.96 \end{bmatrix}$$

(Note $\|\hat{t}_1\|_2 = \sqrt{0.64 + 0.36} = 1$ and $\|\hat{t}_2\|_2 = \sqrt{0.0784 + 0.9216} = 1$.)

1. **Cosine similarity matrix $\hat{V}\hat{T}^\top$:**

$$\hat{v}_1^\top \hat{t}_1 = 0.80, \quad \hat{v}_1^\top \hat{t}_2 = 0.28, \quad \hat{v}_2^\top \hat{t}_1 = 0.60, \quad \hat{v}_2^\top \hat{t}_2 = 0.96$$

2. **Scaled logit matrix $S = 2.0\,\hat{V}\hat{T}^\top$:**

$$S = \begin{bmatrix} 1.60 & 0.56 \\ 1.20 & 1.92 \end{bmatrix}$$

3. **Row-wise losses (image $\to$ text):** Using $-\log \frac{e^a}{e^a + e^b} = \log(1 + e^{b-a})$:
   - Row 1: $\log(1 + e^{0.56 - 1.60}) = \log(1 + e^{-1.04}) = \log(1 + 0.3535) \approx 0.3027$
   - Row 2: $\log(1 + e^{1.20 - 1.92}) = \log(1 + e^{-0.72}) = \log(1 + 0.4868) \approx 0.3966$
   - Average: $\mathcal{L}_{\text{img}\to\text{txt}} = \frac{0.3027 + 0.3966}{2} \approx 0.3497$

4. **Column-wise losses (text $\to$ image):**
   - Column 1: $\log(1 + e^{1.20 - 1.60}) = \log(1 + e^{-0.40}) = \log(1 + 0.6703) \approx 0.5130$
   - Column 2: $\log(1 + e^{0.56 - 1.92}) = \log(1 + e^{-1.36}) = \log(1 + 0.2567) \approx 0.2285$
   - Average: $\mathcal{L}_{\text{txt}\to\text{img}} = \frac{0.5130 + 0.2285}{2} \approx 0.3708$

5. **Total symmetric loss:**

$$\mathcal{L}_{\text{CLIP}} = \frac{0.3497 + 0.3708}{2} \approx 0.3602$$

Inspecting the four directional terms reveals why symmetry matters: image 1 easily prefers caption 1 over caption 2 (row-1 loss $= 0.3027$), yet caption 1 is uncomfortably close to distractor image 2 ($\hat{v}_2^\top \hat{t}_1 = 0.60$), producing the largest individual penalty (column-1 loss $= 0.5130$). A one-sided row loss would under-penalize that confusion.

CLIP delivered a landmark empirical result: evaluated **zero-shot** on ImageNet (converting each class name into a prompt like `"a photo of a {class}"` and selecting the highest cosine similarity), CLIP outperformed a supervised ResNet-50 trained on all 1.2M labeled ImageNet images.

COMMON MISCONCEPTION

You might think that training a vision encoder by predicting the exact caption word-for-word (autoregressive language modeling conditioned on the image) would provide a richer learning signal than CLIP's contrastive ranking. CLIP's own ablation proved the opposite: autoregressive caption prediction is far less compute-efficient because a single image can be described truthfully in countless phrasings, so predicting exact surface wording wastes model capacity compared to distinguishing semantic matches from distractors.

Despite establishing the semantic backbone for multimodal models, CLIP suffers from three structural limitations when reused inside modern VLMs:

1. **Coarse classification bias:** Its global image-caption matching objective targets high-level object recognition, losing fine-grained spatial detail and small text required for optical character recognition (OCR).

2. **Destructive preprocessing:** Fixed 336×336 resizing and center-cropping discards original aspect ratios and slices off border content.
3. **Full-batch softmax coupling:** Every row and column denominator sums over all N examples in the batch, coupling loss quality tightly to massive batch sizes (such as $32{,}768$) and forcing expensive global synchronization across all training workers.

> **KEY IDEA**
> CLIP proved that web-scale contrastive image-caption alignment learns stronger, open-vocabulary visual representations than supervised class labels or generative caption prediction, but its full-batch softmax and 336×336 center-cropping create scaling and fine-detail bottlenecks.

The third limitation—coupling representation quality to a globally synchronized N-way softmax—forces a direct question: can we preserve contrastive image-text alignment while eliminating the batch-wide softmax denominator entirely?

> **✎ Check Your Understanding**
>
> 1. **Quick computation:** Suppose a ViT uses 16×16 patches on a 384×384 cropped RGB image. How many visual patch tokens enter the Transformer, and how many scalar pixel values are in each patch?
>
>   **Answer:** Dividing $384/16 = 24$ along each axis yields $24 \times 24 = 576$ visual tokens, and each RGB patch contains $16 \times 16 \times 3 = 768$ scalar pixel values.
>
> 2. **What would happen if…** you removed the learnable scale $\tau = e^t$ from CLIP's similarity logits $S_{ij} = \tau\, \hat{v}_i^\top \hat{t}_j$ (fixing $\tau = 1$) while keeping L_2 normalization?
>
>   **Answer:** Every logit S_{ij} would be trapped in $[-1, 1]$, capping the maximum possible softmax probability for the true match far below 1 when N is large and preventing the model from sharpening its predictions.

Section 3: Decoupling the Contrastive Objective — SigLIP and SigLIP-2

In CLIP, every pair (i, j) interacts with every other pair in its row and column through the softmax denominators $\sum_{k=1}^{N} e^{S_{ik}}$ and $\sum_{k=1}^{N} e^{S_{kj}}$. Computing exact gradients therefore requires gathering embeddings or log-sum-exp statistics across the entire global batch of N examples. **SigLIP** (Sigmoid Loss for Language-Image Pre-Training, 2023) eliminates that global coupling with one conceptual change: instead of framing alignment as N-way multiclass classification across the batch, treat every single `(image, text)` pair (i, j) as an independent **binary classification** problem—aligned ($z_{ij} = +1$) or not aligned ($z_{ij} = -1$).

> **DEFINITION**
> **Sigmoid Loss for Language-Image Pre-Training (SigLIP):** A pairwise contrastive objective that evaluates every `(image, text)` pair independently with binary logistic (sigmoid) loss using a learnable temperature and negative bias, removing batch-wide softmax normalization.

Given L_2-normalized image and text embeddings $\hat{v}_i, \hat{t}_j \in \mathbb{R}^d$, define the binary label $z_{ij} = +1$ when $i = j$ (matching pair) and $z_{ij} = -1$ when $i \neq j$ (mismatched pair), along with a learnable positive scale $\tau = e^{t'}$ and a learnable scalar bias b. The SigLIP loss is:

$$\mathcal{L}_{\text{SigLIP}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\log\frac{1}{1+\exp\bigl(-z_{ij}\,(\tau\,\hat{v}_i^\top\hat{t}_j + b)\bigr)}$$

Breaking this formula apart hierarchically from the innermost expression outward:

1. **Affinely shifted similarity logit ($u_{ij} = \tau\,\hat{v}_i^\top\hat{t}_j + b$):** As in CLIP, $\tau\,\hat{v}_i^\top\hat{t}_j$ scales the cosine similarity between image i and caption j. Unlike CLIP—where any constant additive shift cancels inside softmax—binary sigmoid classification depends on the absolute threshold 0, so adding a learnable bias b sets the decision boundary between matching and non-matching pairs.
2. **Signed margin ($z_{ij}\,u_{ij}$):** Multiplying by $z_{ij} \in \{+1, -1\}$ makes the margin positive whenever the pair is classified correctly (large positive u_{ij} on the diagonal, large negative u_{ij} off the diagonal).
3. **Sigmoid probability ($\sigma(z_{ij}u_{ij}) = \frac{1}{1+\exp(-z_{ij}u_{ij})}$):** Maps the signed margin into $(0, 1)$, giving the predicted probability of the true binary label for pair (i, j) completely locally, with zero dependence on any other entry (i, k) or (k, j).
4. **Negative log-likelihood ($-\log\sigma(z_{ij}u_{ij})$):** Computes the binary cross-entropy loss of pair (i, j).
5. **Double sum normalized by N ($-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}[\dots]$):** Sums the binary losses across all N^2 pairs and divides by N (the number of positive pairs).

As a single conceptual unit, $\mathcal{L}_{\text{SigLIP}}$ scores every cell of the $N \times N$ grid independently as a yes/no match detector, making the total batch loss a pure additive sum over pairs.

Why must the learnable bias b be initialized to a **negative value** ($b < 0$)? In any batch of size N, there are only N positive pairs on the diagonal but $N^2 - N$ negative pairs off the diagonal. If $b = 0$ at initialization when random embeddings have cosine similarities near zero, the $N^2 - N$ negative pairs flood the early gradients and destabilize training. Setting $b < 0$ shifts all initial logits negative so negative pairs already incur tiny loss while positive pairs supply clean, balanced gradients.

WORKED EXAMPLE

Quantify the gradient imbalance at initialization for a batch of $N = 100$ pairs (100 positive pairs on the diagonal and $N^2 - N = 9{,}900$ negative pairs off the diagonal). Assume untrained embeddings are nearly orthogonal so $\hat{v}_i^\top\hat{t}_j \approx 0$ for every (i, j), leaving $u_{ij} = b$.

Recall that $\frac{\partial}{\partial b}\bigl[-\log\sigma(z_{ij}b)\bigr] = -z_{ij}\bigl(1 - \sigma(z_{ij}b)\bigr)$.

1. **Zero bias ($b = 0$):** Here $\sigma(0) = 0.5$ and $-\log(0.5) = \ln 2 \approx 0.693$ for every pair.
   - Positive-pair gradient sum (w.r.t. b): $100 \times (-0.5) = -50$.
   - Negative-pair gradient sum (w.r.t. b): $9{,}900 \times (+0.5) = +4{,}950$.
   - The negative pairs overwhelm the positive pairs by a $99:1$ ratio, and the loss starts at $(100 + 9{,}900) \times 0.693/100 = 69.3$.
2. **Negative initial bias ($b = -5$):**
   - For each positive pair ($z_{ij} = +1$): $\sigma(-5) = \frac{1}{1+e^5} \approx 0.00669$, loss ≈ 5.007, gradient $\approx -(1 - 0.00669) = -0.9933$. Sum over 100 positives: -99.33.
   - For each negative pair ($z_{ij} = -1$): $\sigma(+5) \approx 0.99331$, loss ≈ 0.00671, gradient $\approx +(1 - 0.99331) = +0.00669$. Sum over $9{,}900$ negatives: $+66.26$.
   - The total loss drops to $(100 \times 5.007 + 9{,}900 \times 0.00671)/100 \approx 5.67$, and the positive and negative gradient magnitudes (99.33 versus 66.26) are balanced on the same order of magnitude.

Initializing $b < 0$ directly prevents the $N^2 - N$ off-diagonal negatives from drowning out the N diagonal positives at the start of training.

In PyTorch, SigLIP is just as concise as CLIP:

```
z_img = F.normalize(img_enc(I), dim=-1)
z_txt = F.normalize(txt_enc(T), dim=-1)

# [N, N] pairwise scores
logits = (z_img @ z_txt.T) * t + b

# +1 on diagonal, -1 off-diagonal
labels = 2 * torch.eye(N) - 1

loss = -F.logsigmoid(
    labels * logits
).sum() / N
```

The expression `2 * torch.eye(N) - 1` constructs an $N \times N$ matrix containing $+1$ on the diagonal and -1 everywhere else, and `-F.logsigmoid(labels * logits).sum() / N` evaluates the exact pairwise binary loss.

Because the double sum over (i, j) contains no cross-row or cross-column normalization, SigLIP unlocks **chunked ring evaluation** across distributed devices without ever materializing the full $N \times N$ similarity matrix.

DEFINITION

Chunked ring evaluation: A distributed strategy for computing SigLIP's pairwise loss in which each device holds a local shard of image embeddings and passes shards of text embeddings around a ring, accumulating local block losses additively without global softmax synchronization or full $N \times N$ matrix allocation.

Trace the three-step **Chunked Ring Evaluation** diagram across three devices (**Dev 0**, **Dev 1**, **Dev 2**), where each device starts with its own image shard (I_0, I_1, I_2) and corresponding text shard (T_0, T_1, T_2):

- **Step 1 (local + positives):** Each device computes only its local block product—Dev 0 computes $I_0 \cdot T_0^\top$, Dev 1 computes $I_1 \cdot T_1^\top$, and Dev 2 computes $I_2 \cdot T_2^\top$ (highlighted boxes containing the diagonal positive pairs plus local off-diagonal negatives)—evaluates the sigmoid loss on that small block, and adds it to a running scalar sum.
- **Ring rotation:** Each device keeps its image shard stationary and rotates its text chunk T_k to the neighboring device in the ring.
- **Step 2 (negatives):** Dev 0 now computes $I_0 \cdot T_1^\top$, Dev 1 computes $I_1 \cdot T_2^\top$, and Dev 2 computes $I_2 \cdot T_0^\top$. Every entry in these off-diagonal blocks is a negative pair ($z_{ij} = -1$). Each device adds its block's loss and discards the block immediately.
- **Step 3 (negatives):** After one more ring rotation of T_k, Dev 0 computes $I_0 \cdot T_2^\top$, Dev 1 computes $I_1 \cdot T_0^\top$, and Dev 2 computes $I_2 \cdot T_1^\top$, accumulating the final negative losses.

Across all three steps, every single (i, j) entry of the full $N \times N$ grid is visited exactly once, yet no device ever stores more than a $(N/3) \times (N/3)$ block in memory, and no global denominator synchronization is ever needed.

COMMON MISCONCEPTION

You might think that CLIP could simply run this same chunked ring loop and sum each block's cross-entropy loss directly. That fails because softmax cross-entropy is non-additive across columns: to compute the exact probability and gradient for row i in Step 1, Dev 0 must already know the full-row denominator $\sum_{j=1}^{N} e^{S_{ij}}$ across all text shards T_0, T_1, T_2, requiring global synchronization across devices before any block's loss or gradient can be finalized.

Alongside its simpler loss, SigLIP paired this objective with a major data upgrade—the **WebLI** dataset:

- $O(\text{billion})$ `(image, text)` pairs scraped from the Internet.
- **Automatic OCR** run on images to extract text appearing inside the images themselves, directly attacking CLIP's weakness on embedded text and documents.

- Quality filtering retaining only the **top 10%** highest-quality examples.
- Multilingual coverage spanning **100 languages**.

In batch-size scaling, decoupling the loss from batch-wide softmax pays off at both ends of the spectrum: SigLIP outperforms CLIP at smaller batch sizes ($< 16\text{K}$), scales cleanly up to a **1M** batch size via chunked evaluation (though **32K** is already sufficient for peak performance), and trains substantially faster with far fewer devices than CLIP by eliminating global softmax synchronization.

The following table summarizes the shift from CLIP to SigLIP and its successor **SigLIP-2** (2025), which preserves the same core SigLIP architecture while upgrading training signals and visual representation quality:

Dimension	CLIP (2021 baseline)	SigLIP / SigLIP-2 (2023–2025)
Objective	Symmetric N-way softmax cross-entropy	Pairwise binary sigmoid loss (learnable τ, b)
Global sync	Full-batch softmax over all N examples	Additive per-chunk loss; no full $N \times N$ gather
Batch size	Needs very large batches (e.g., $32{,}768$)	Beats CLIP below 16K; scales to 1M (32K sufficient)
Pretraining data	400M private pairs (OpenCLIP: LAION-5B)	WebLI: $O(\text{B})$ pairs, auto-OCR, top 10%, 100 languages
VLM usage	Early VLMs (LLaVA, Qwen-VL)	Modern VLMs (LLaVA-OneVision, Qwen3-VL)

KEY IDEA

Replacing CLIP's full-batch softmax with independent pairwise sigmoid classification—combined with OCR-enriched, top-10% filtered WebLI data—eliminates global softmax synchronization, enables chunked ring evaluation, and made SigLIP and SigLIP-2 the standard vision encoders for modern VLMs.

We now have continuous vision encoders that produce strong, OCR-aware visual representations. However, a dual encoder can only score image-text similarity; it cannot answer open-ended questions, follow instructions, or reason over a visual scene. How do we connect a continuous vision encoder to a pretrained language model?

✎ Check Your Understanding

1. **Yes or No:** In SigLIP's chunked ring evaluation across P devices, does any device need to wait for all P ring steps to finish before knowing the exact loss value of an individual pair (i, j) in its current block? Justify your answer.

   Answer: No. Because SigLIP evaluates every pair (i, j) with an independent binary sigmoid loss $-\log \sigma(z_{ij}(\tau \hat{v}_i^\top \hat{t}_j + b))$, each cell's loss depends only on $\hat{v}_i$ and $\hat{t}_j$ and requires no global denominator from other blocks.

2. **What would happen if…** you trained SigLIP with the learnable bias b initialized to a large *positive* number (e.g., $b = +5$) instead of a negative number?

   Answer: All $N^2 - N$ negative pairs would start with strongly wrong positive logits ($z_{ij} u_{ij} = -5$), producing enormous initial loss and gradients dominated almost entirely by the mismatched pairs rather than balanced learning between matches and non-matches.

Section 4: Connecting Vision to Language — The Modular VLM Blueprint and the Resolution Bottleneck

Pretrained LLMs already possess broad world knowledge and instruction-following ability, while a pretrained ViT (such as CLIP or SigLIP) already maps pixels into rich visual patch vectors. The two models, however, live in incompatible vector spaces of different dimensions and geometries. Early multimodal architectures bridged them with heavy cross-attention modules (such as Flamingo or Q-Former). In 2023, **LLaVA** (a historical baseline) demonstrated a far simpler solution that established the canonical **modular VLM** blueprint: **Vision Encoder + Projector + LLM**.

> **DEFINITION**
> **Modular Vision-Language Model (VLM):** An understanding-oriented multimodal architecture that connects a continuous pretrained vision encoder to a pretrained LLM backbone through a lightweight projector or adapter, injecting projected visual features directly as input tokens alongside text embeddings.

Trace the LLaVA pipeline from left to right across its two parallel input paths:

- **Image path (top):** An input image X_v passes through a **frozen CLIP ViT-L/14** encoder. Recall from Section 2 that CLIP normally ends with an attention pooling layer that collapses all 576 patches into a single vector; LLaVA *drops that final pooling layer* so the encoder outputs the full grid of spatial patch embeddings Z_v. A single **linear projection matrix** W transforms those visual features into the LLM's word-embedding space:

$$H_v = W \cdot Z_v$$

- **Text path (bottom):** The user's text instruction X_q is converted through the LLM's standard lookup table into text token embeddings H_q.
- **Fusion and generation (right):** The projected visual tokens H_v and text tokens H_q are concatenated into one unified sequence and fed into a **Vicuna LM** (LLaMA fine-tuned on ShareGPT), which autoregressively generates the text answer X_a.

> **WORKED EXAMPLE**
> Compute the matrix dimensions, projector parameter count, and input sequence length for LLaVA when pairing CLIP `ViT-L/14@336px` (patch feature dimension $d_{\text{vis}} = 1{,}024$) with a Vicuna backbone of hidden dimension $d_{\text{LM}} = 4{,}096$, given a 40-token text question:
>
> 1. **Visual grid shape:** From Section 2, 336×336 sliced into 14×14 patches yields $24 \times 24 = 576$ spatial tokens, so $Z_v \in \mathbb{R}^{576 \times 1024}$ (or 1024×576 column-wise).
> 2. **Projector size:** Mapping each $1{,}024$-dimensional patch vector into $\mathbb{R}^{4096}$ requires a single linear matrix $W \in \mathbb{R}^{4096 \times 1024}$, containing $4{,}096 \times 1{,}024 = 4{,}194{,}304 \approx 4.19\text{M}$ parameters—less than 0.1% of a 7B-parameter LLM.
> 3. **Combined LLM sequence:** Multiplying gives $H_v \in \mathbb{R}^{576 \times 4096}$, which concatenates with $H_q \in \mathbb{R}^{40 \times 4096}$ to form a 616×4096 sequence entering Layer 0 of Vicuna.
>
> A connector with only 4.19M parameters suffices to translate the visual patch grid into tokens the LLM can attend over directly.

Architecture alone was not enough: web `(image, caption)` pairs teach a model to emit terse descriptions, not to follow multi-turn visual instructions or reason about a scene. LLaVA solved the data shortage with **synthetic visual instruction tuning**, forming a three-box data generation pipeline:

DEFINITION

Visual instruction tuning: Fine-tuning a vision-language model on multimodal question-answer pairs, detailed descriptions, and multi-turn conversations—often bootstrapped synthetically by prompting a strong text-only LLM with existing image annotations—so the model follows open-ended visual instructions.

1. **MS COCO Annotations:** Start from existing image datasets that already include human-labeled bounding boxes and Mechanical Turk captions.
2. **Prompt Text-Only GPT-4:** Feed those textual bounding boxes and captions into text-only GPT-4 (which never sees the raw image) and prompt it to generate rich visual Q&A, detailed descriptions, and multi-turn conversations.
3. **Pair with Original Images:** Attach the generated text conversations back to the original MS COCO images, producing **158K multimodal examples** for visual instruction tuning.

Bootstrapping multimodal data from structured text annotations plus a strong LLM became a standard recipe across open VLMs.

LLaVA trained this architecture in **two stages**:

Training Stage	Vision Encoder	Projector W	Language Model	Purpose
Stage 1 (Alignment)	Frozen	**Train**	Frozen	Align visual features to LM space without wrecking LM weights
Stage 2 (Fine-tuning)	Frozen	**Train**	**Train**	End-to-end multimodal instruction following

COMMON MISCONCEPTION

You might think that freezing the language model in Stage 1 wastes compute and that unfreezing both W and the LLM from step 0 would converge faster. That breaks down because at step 0, W is randomly initialized, so the projected visual tokens $H_v = W \cdot Z_v$ enter the LLM as unstructured noise; backpropagating large, noisy gradients through the pretrained LLM weights immediately degrades its language reasoning before the projector has learned even a rough alignment.

LLaVA proved that a simple linear projection plus synthetic instruction data unlocks visual reasoning in an LLM. Almost immediately, however, practical tasks exposed a severe flaw inherited from Section 2: **the resolution bottleneck**. Resizing and center-cropping every image to 336×336 pixels blurs small characters and destroys fine visual details needed for OCR, dense documents, and wide charts.

To bypass fixed encoder resolution without retraining the vision encoder from scratch, LLaVA 1.5 introduced—and **LLaVA-OneVision** extended—a tiling strategy called **AnyRes**.

DEFINITION

AnyRes: A multi-tile preprocessing method that splits a high-resolution image into an $a \times b$ grid of crops matching the vision encoder's native resolution alongside a downsampled global overview image, encodes each tile separately with the ViT, concatenates all resulting tokens, and applies bilinear interpolation pooling if the total token count exceeds a set budget.

Follow the AnyRes dataflow diagram from top-left to bottom:

1. **High-Res Image (e.g., wide chart / OCR):** Enters at its original resolution and aspect ratio.
2. **Dual split:** One arrow resizes the entire image into a single **Base (global)** thumbnail preserving coarse layout; a second arrow slices the full image into an $a \times b$ **grid of encoder-sized tiles** (depicted as a 2×3 grid of 6 tiles) preserving local high-resolution detail.

3. **Encode + Concatenate:** Each tile and the global base pass independently through the ViT, and all patch tokens are concatenated into one visual sequence.
4. **Budget control:** If the concatenated sequence contains too many tokens for the LLM's budget, the spatial feature grids are downsampled using **bilinear interpolation pooling**.

> **WORKED EXAMPLE**
> Apply AnyRes to a 1008×672 high-resolution document page using a 336×336 vision encoder that outputs a $24 \times 24 = 576$ token grid per crop:
>
> 1. **Tile grid selection:** Because $1008/336 = 3$ columns and $672/336 = 2$ rows, AnyRes partitions the document into $a \times b = 2 \times 3 = 6$ native-resolution tiles without discarding a single border pixel, plus 1 global base thumbnail (7 crops total).
> 2. **Unpooled token count:** Encoding all 7 crops produces $7 \times 576 = 4{,}032$ visual tokens.
> 3. **Bilinear interpolation pooling:** Suppose the token budget requires shrinking each tile's 24×24 spatial grid by a factor of 2 along height and width via bilinear interpolation to $12 \times 12 = 144$ tokens while keeping the global base at 576 tokens. The total visual sequence drops to:
>
> $$576 + 6 \times 144 = 576 + 864 = 1{,}440 \text{ tokens}$$
>
> Every word on the page remains legible at 336 px per tile during ViT encoding, yet the LLM receives nearly $3\times$ fewer tokens than the unpooled grid.

LLaVA-OneVision (2024, open weights and data) generalized this modular recipe from single images to three visual regimes simultaneously—**single images**, **multiple images**, and **video**—using three architectural upgrades over original LLaVA:

- **Vision encoder: SigLIP**, extracting spatial grid features both *before and after* the final Transformer layer.
- **Projector:** A **2-layer MLP** instead of a single linear matrix.
- **Text decoder: Qwen-2 72B**.

When a single model trains jointly on images, multi-image sets, and multi-frame videos, token lengths can swing wildly across modalities. LLaVA-OneVision therefore enforces **token-budget balancing** across the three input types so that all three occupy roughly similar sequence lengths—preventing any one modality from starving or overwhelming the context window. Picture three side-by-side panels representing the trade-off between spatial resolution and frame count:

1. **Single Image:** Allocated **higher resolution** via an AnyRes multi-tile grid (e.g., 2×3 tiles), spending the full visual token budget on fine spatial detail.
2. **Multiple Images:** Allocated **base resolution each** (1 base crop per image across several images), trading per-image tiling for multi-image breadth.
3. **Video:** Allocated **lower resolution per frame** (many frames whose spatial grids are heavily bilinearly pooled), spending the same token budget across time rather than high per-frame pixel count.

Beyond architecture, LLaVA-OneVision highlighted two operational principles of modular VLMs:

- **Data philosophy — quality over quantity:** Most engineering effort in modular VLMs goes into curating synthesized, task-specific datasets rather than indiscriminately scraping raw web data.
- **Training philosophy — easy-to-hard curriculum:** Training progresses deliberately from easier alignment tasks, to single-image knowledge acquisition, to mixed single-image, multi-image, and video reasoning.

Once the vision-language interface is shared across these regimes, LLaVA-OneVision exhibited striking **emergent cross-modal transfer**:

- Diagram and chart understanding trained on **single images** generalized directly to **multi-image** comparison.

- **Single-image OCR** combined with **multi-image relational reasoning** transferred to **GUI-based agents** navigating screen sequences.
- **Visual prompting** (such as circled regions drawn on **single images**) transferred directly to **video** without video-specific visual-prompting supervision.

> **KEY IDEA**
> LLaVA established the canonical modular VLM pipeline (Vision Encoder + Projector + LLM) trained in stages with synthetic instruction data; extending it with AnyRes tiling, modality-balanced token budgets, and easy-to-hard curricula unlocks strong cross-modal transfer across images, multi-image sets, and video.

Even with AnyRes, however, modular VLMs still carry two geometric flaws. First, slicing an image into separate square tiles splits objects across artificial seams and forces each tile to be encoded in isolation. Second, concatenating a 2D patch grid (or 3D video volume) into a 1D token stream fed into standard 1D positional encodings blinds the LLM to true vertical and temporal adjacency. Addressing those structural flaws is the exact progression traced by the Qwen-VL family.

> **✎ Check Your Understanding**
>
> 1. **Fill in the blank:** In LLaVA's two-stage training recipe, Stage 1 trains only the _____ while keeping both the _____ and the _____ frozen.
>
>   **Answer:** **Projector** (W); **vision encoder**; **language model**. This aligns visual features to the LLM's embedding space before unfreezing the LLM in Stage 2.
>
> 2. **What would happen if…** LLaVA-OneVision applied full multi-tile AnyRes to every frame of a 64-frame video instead of bilinearly pooling each frame?
>
>   **Answer:** Each frame would expand into thousands of visual tokens, causing a single video to consume tens or hundreds of thousands of tokens and completely overwhelm the context window compared to single-image and multi-image examples.

Section 5: Native Dynamic Resolution, 3D Positional Geometry, and Deep Fusion (The Qwen-VL Progression)

The progression from Qwen-VL (2023) to Qwen2-VL (2024) to Qwen3-VL (late 2025) resolves, one by one, every geometric and architectural limitation left open by fixed-crop and tiled VLMs.

Qwen-VL (2023 Baseline): Compression Adapter and Spatial Grounding

The first **Qwen-VL** (2023) paired an **OpenCLIP ViT** (14×14 patches) with a **1-layer cross-attention resampler** equipped with 2D positional encodings.

> **DEFINITION**
> **Cross-attention resampler:** A learned adapter layer in which a fixed number of learned query vectors cross-attend over all ViT patch embeddings, compressing any image into a constant-length sequence of visual tokens (such as 256 tokens in Qwen-VL).

Qwen-VL also introduced special grounding tokens—`<img>`, `<box>`, and `<ref>`—allowing the language model to emit exact bounding-box coordinates inline as text for object localization. Unlike original LLaVA (which kept the vision encoder frozen throughout), Qwen-VL introduced a **3-stage training schedule** that unfroze the vision encoder during early

alignment and multi-task training:

Stage	Data & Resolution	Vision	Adapter	LM
Stage 1	Large-scale, lower-quality web pairs	**Train**	**Train**	Frozen
Stage 2	Higher-quality task-specific data + higher resolution	**Train**	**Train**	**Train**
Stage 3	Instruction tuning data	Frozen	**Train**	**Train**

Unfreezing the vision encoder in Stages 1 and 2 improved visual alignment significantly. Yet forcing every image—no matter how large or text-dense—through a fixed **256-token bottleneck** still strangled fine-grained reading and document OCR.

Qwen2-VL (2024): Native Dynamic Resolution and Spatial Merging

Qwen2-VL (2024) abandoned both fixed-length resamplers and AnyRes-style square tiling in favor of **native dynamic resolution**.

> **DEFINITION**
> **Native dynamic resolution:** Encoding every image directly at its natural aspect ratio and patch count—without forced square cropping or slicing into independent tiles—so the number of visual tokens scales proportionally to image area.

> **DEFINITION**
> 2×2 **spatial merge:** A lightweight patch-merger module placed between the vision encoder and the LLM that combines each adjacent 2×2 block of ViT patch embeddings into a single visual token, reducing visual sequence length by $4\times$ while preserving 2D neighborhood structure.

Trace the Qwen2-VL pipeline from left to right:

1. **Variable-size images (Small or Native Aspect):** Images of different sizes and aspect ratios enter directly without forced square cropping.
2. **Dynamic ViT (675M parameters):** Initialized from DFN and operating on 14×14 pixel patches, this ViT processes variable-length 2D patch grids natively, so every patch attends globally across the entire image rather than being isolated inside a separate tile.
3. 2×2 **Spatial Merge:** Every 2×2 cluster of four neighboring patch tokens merges into one combined token, cutting the visual token count by $4\times$.
4. **Qwen2 LM:** Receives a variable-length visual token stream wrapped by boundary tokens.

For video, Qwen2-VL samples at **2 frames per second** and caps total video length at **16,384 tokens**. Its **3-stage training** trains (1) the visual encoder only, (2) all parameters jointly, and (3) the LM on instruction-following datasets.

> **WORKED EXAMPLE**
> Verify how native dynamic resolution and 2×2 spatial merging determine visual token counts for two differently shaped inputs using a ViT with 14×14 patches:
>
> 1. **Square 224×224 crop:**
>   - Patch grid: $(224/14) \times (224/14) = 16 \times 16 = 256$ patches.
>   - After 2×2 spatial merge: $(16/2) \times (16/2) = 8 \times 8 = 64$ merged visual tokens.
>   - Adding 2 vision boundary tokens (start and end delimiters) yields $64 + 2 = 66$ tokens.
> 2. **Wide 896×224 banner (4:1 aspect ratio):**

- Patch grid: $(896/14) \times (224/14) = 64 \times 16 = 1{,}024$ patches.
- After 2×2 spatial merge: $(64/2) \times (16/2) = 32 \times 8 = 256$ merged visual tokens.
- Adding 2 boundary tokens yields $256 + 2 = 258$ tokens.

Both the square image and the $4:1$ banner keep their exact native geometry with zero cropped borders, and the $4\times$ larger banner receives exactly $4\times$ as many merged visual tokens (256 versus 64).

Multimodal Rotary Position Embedding (MRoPE)

Native dynamic resolution preserves the 2D grid inside the ViT, but once the merged tokens enter the LLM, a positional-encoding problem surfaces. Recall that standard **1D RoPE** encodes a single scalar sequence position m by rotating consecutive pairs of query and key dimensions by angles $m\theta_k$. When a 2D image grid is flattened row-by-row into a 1D sequence, horizontally adjacent patches sit 1 step apart, whereas vertically adjacent patches sit a full row-width W_{grid} apart —making vertical neighbors look arbitrarily distant and breaking 2D spatial intuition.

Qwen2-VL solved this with **Multimodal Rotary Position Embedding (MRoPE)**.

> **DEFINITION**
> **Multimodal Rotary Position Embedding (MRoPE):** An extension of RoPE that partitions the rotary embedding channels into three coordinate axes—temporal (t), height (h), and width (w)—assigning (t, h, w) coordinates to every token so 1D text, 2D images, and 3D space-time video share one unified positional scheme.

Visualize the MRoPE vector bar divided into three contiguous colored segments—**Temporal (t)**, **Height (h)**, and **Width (w)**—and observe how coordinates are assigned for each modality:

- **Text tokens:** Receive coordinates `(0, 0, 0)`, `(1, 1, 1)`, `(2, 2, 2)`, … Setting $t = h = w$ rotates every channel pair by the exact same scalar position m, making MRoPE **mathematically identical to standard 1D RoPE** and guaranteeing seamless backward compatibility with pretrained text-only LLM weights.
- **Image tokens:** Following a 3-token text prefix, image patches receive `(3, 0, 0)`, `(3, 0, 1)`, …, `(3, h, w)`. The temporal index t stays constant across the entire image while (h, w) records the exact 2D patch grid row and column.
- **Video frames:** Receive `(t, h, w)` where t increments per frame while (h, w) tracks spatial position within each frame —unifying 1D text, 2D images, and 3D space-time video in one mechanism.

Qwen3-VL (Late 2025): Four Surgical Architectural Upgrades

Qwen3-VL (late 2025) illustrates the mature open modular VLM recipe, combining a **SigLIP-2** vision encoder with Qwen-3 dense and sparse MoE language backbones scaling up to **235B-A22B** (235B total parameters, 22B active per token) and native multimodal context up to **256K tokens**. On top of this scale, Qwen3-VL introduced **four targeted architectural fixes**.

1. Interleaved MRoPE and Explicit Video Timestamps

Block MRoPE in Qwen2-VL harbored a subtle frequency-band flaw. Recall that in RoPE, channel pairs are ordered across a geometric frequency spectrum $\theta_k = b^{-2k/d}$: the leftmost channels rotate at **high frequency** (resolving fine local detail and immediate neighbors), whereas the rightmost channels rotate at **low frequency** (resolving long-range global distance without wrapping around).

Compare the two horizontal channel-layout bars side by side, labeled `← High frequency (fine local detail)` on the left and `Low frequency (long-range) →` on the right:

- **Block MRoPE (Qwen2-VL):** Lays out channels in three contiguous blocks: `[t t t t | w w w w | h h h h]`. Each axis is trapped inside a narrow frequency band: temporal index t receives *only* high frequencies, width w receives only mid-range frequencies, and height h receives *only* low frequencies.

- **Interleaved MRoPE (Qwen3-VL):** Cycles axes across the entire spectrum: `[t w h | t w h | t w h | t w h]`. All three axes (t, w, h) now span both high-frequency (fine local) and low-frequency (long-range) bands evenly.

DEFINITION

Interleaved MRoPE: A channel layout for Multimodal RoPE that alternates temporal, width, and height coordinate assignments (`[t, w, h, t, w, h, ...]`) across the rotary frequency spectrum so every axis receives balanced short-range and long-range positional resolution.

COMMON MISCONCEPTION

You might think that dividing the rotary embedding dimensions into three equal thirds—one-third contiguous block for t, one-third for w, and one-third for h—gives all three spatial-temporal axes equal representational quality. That fails because RoPE channels are not interchangeable: rotary frequencies decay exponentially with channel index, so contiguous blocks starve the vertical axis h of high-frequency local precision and starve the temporal axis t of low-frequency long-range capacity.

WORKED EXAMPLE

Compare the rotary frequencies assigned to (t, w, h) in a head of dimension $d_k = 24$ (12 rotary channel pairs, indexed $k = 0, 1, \ldots, 11$) using standard base $b = 10{,}000$, where $\theta_k = 10{,}000^{-2k/24} = 10^{-k/3}$:

Pair k	0	1	2	3	4	5	6	7	8	9	10	11
θ_k	1.00	0.464	0.215	0.100	0.0464	0.0215	0.0100	0.00464	0.00215	0.00100	0.000464	0.000215

1. **Block MRoPE (`t t t t | w w w w | h h h h`):**
   - Temporal t gets $k \in \{0, 1, 2, 3\}$, so $\theta \in [0.100, 1.00]$. Even on its slowest channel ($\theta_3 = 0.100$ rad/frame), a 100-frame video rotates through $100 \times 0.100/(2\pi) \approx 1.59$ full revolutions, aliasing long-range temporal order.
   - Height h gets $k \in \{8, 9, 10, 11\}$, so $\theta \in [0.000215, 0.00215]$. Two vertically adjacent rows ($\Delta h = 1$) differ by at most 0.00215 rad $\approx 0.12°$, making neighboring rows nearly indistinguishable in fine-grained OCR.
2. **Interleaved MRoPE (`t w h | t w h | t w h | t w h`):**
   - t gets $k \in \{0, 3, 6, 9\}$: $\theta \in [0.00100, 1.00]$.
   - w gets $k \in \{1, 4, 7, 10\}$: $\theta \in [0.000464, 0.464]$.
   - h gets $k \in \{2, 5, 8, 11\}$: $\theta \in [0.000215, 0.215]$.

With zero extra parameters or compute, interleaving gives every single axis three full orders of magnitude of frequency coverage—preserving crisp local discrimination ($\Delta h = 1$ now rotates by 0.215 rad $\approx 12.3°$) and alias-free long-range tracking (100 frames on $\theta_9 = 0.001$ rotates by only 0.1 rad).

2. Explicit Video Timestamps

Even with Interleaved MRoPE, relying solely on integer positional IDs to represent elapsed time breaks down whenever frame sampling rates vary across videos. Qwen3-VL therefore inserts **explicit video timestamp markers** as separate text tokens ahead of video frames, grounding temporal queries directly in wall-clock time.

3. DeepStack Cross-Layer Visual Feature Injection

Standard modular VLMs inject visual tokens only at **LLM Layer 0**. That design forces the final layer of the ViT to summarize every low-level edge, character stroke, and high-level object concept into a single output representation.

DEFINITION

DeepStack: A cross-layer multimodal adapter architecture that projects intermediate representations from early, middle, and top layers of the vision encoder and injects them into multiple corresponding layers of the LLM decoder.

Picture the **DeepStack** architecture as two vertical stacks connected by horizontal bridges:

- On the left, the **SigLIP-2 Vision Encoder** flows upward through **Early ViT Layers**, **Mid ViT Layers**, and **Top ViT Layers**.
- On the right, the **LLM Decoder Layers** flow upward through **LLM Layer 0 (in)**, **LLM Layer** k_1, and **LLM Layer** k_2.
- Between them sit three dedicated projectors: **Proj 0** injects Early ViT features into LLM Layer 0, **Proj 1** injects Mid ViT features into LLM Layer k_1, and **Proj 2** injects Top ViT features into LLM Layer k_2.

Because intermediate visual features are fused into the existing visual token positions across depth rather than appended as extra tokens, DeepStack preserves both fine-grained low-level cues (edges, OCR characters) and high-level semantics *without inflating context length*.

4. Square-Root-Normalized Per-Token Loss and Staged Context Extension

Training jointly on text, images, and long videos introduces a severe optimization imbalance: video sequences contain vastly more tokens than text examples while carrying far lower information density per token. Under standard per-token cross-entropy loss (where every token in a batch contributes equal weight), a few long videos overwhelm gradient updates and degrade text capability.

DEFINITION

Square-root-normalized per-token loss: A sequence-level loss weighting scheme that downweights very long multimodal sequences proportionally to $1/\sqrt{L}$ (making a sequence of length L contribute proportionally to $\sqrt{L}$ rather than L) to balance low-density video tokens against dense text tokens during joint training.

WORKED EXAMPLE

Compare three loss-weighting rules on a training batch containing one 100-token text example and one $10{,}000$-token video example, assuming both have an average per-token cross-entropy loss of $\bar{\ell} = 2.0$:

1. **Standard per-token loss (each token weighted by 1):**
   - Text total: $100 \times 2.0 = 200$.
   - Video total: $10{,}000 \times 2.0 = 20{,}000$.
   - Video-to-text gradient influence ratio: $20{,}000/200 = 100:1$. The single video completely dominates the parameter update.
2. **Flat per-sequence averaging (each token weighted by $1/L$):**
   - Text total: 2.0; Video total: 2.0.
   - Ratio: $1:1$. Now a $10{,}000$-token video receives no more total gradient budget than a short sentence, severely under-training long-context reasoning.
3. **Square-root-normalized per-token loss (each token weighted by $1/\sqrt{L}$, so a sequence contributes $\frac{1}{\sqrt{L}} \sum_{m=1}^{L} \ell_m = \sqrt{L}\,\bar{\ell}$):**
   - Text total: $\sqrt{100} \times 2.0 = 20$.
   - Video total: $\sqrt{10{,}000} \times 2.0 = 200$.
   - Ratio: $200/20 = 10:1$—a tenfold reduction in video dominance compared to standard per-token weighting, keeping text and long-video optimization balanced while still giving longer sequences meaningful weight.

Qwen3-VL combines these four upgrades with a **4-stage pretraining** and **3-step post-training** pipeline mirroring progressive text-LLM context extension:

- **Stage 0:** Train adapter only (align vision encoder and LM).
- **Stage 1:** Train all parameters at **8K context**.
- **Stage 2:** Train all parameters at **32K context**.
- **Stage 3:** Train all parameters at **256K context** (scaling cleanly to long videos and multi-page documents).
- **Post-Training:** (1) **Long-CoT SFT**, (2) **Distillation**, and (3) **RL**.

Evolution of the Modular VLM Template and Late-2026 Backbones

Putting all five modular VLM milestones side by side shows a clear architectural trajectory away from fixed crops and fixed-length resamplers toward native dynamic resolution, 3D rotary positions, deep fusion, and sparse MoE backbones:

Model	Vision Encoder	Resolution	Adapter	Pos. / Backbone
LLaVA (2023 baseline)	CLIP ViT-L/14	Fixed 336×336 crop	Linear W	1D RoPE / Vicuna (LLaMA)
Qwen-VL (2023 baseline)	OpenCLIP ViT/14	Fixed crop, higher in Stage 2	1-layer cross-attn $\rightarrow$ 256 tokens	2D pos in adapter; `<box>`, `<ref>`
LLaVA-OneVision (2024)	SigLIP (2-layer grid)	AnyRes tiles + bilinear pool	2-layer MLP	Qwen-2 72B (img / multi / vid)
Qwen2-VL (2024)	675M ViT (DFN init)	Native dynamic + 2×2 merge	Patch merger	MRoPE / Qwen2 (vid 2 fps, $\leq 16\text{K}$)
Qwen3-VL (late 2025)	SigLIP-2	Native dynamic + timestamps	DeepStack (multi-layer)	Interleaved MRoPE / Qwen-3 MoE (235B-A22B, 256K)

Because images and video inflate sequence lengths so aggressively, modern multimodal systems depend heavily on advances in the underlying language backbone. In late-2026 frontier open backbones, three backbone innovations directly relieve the visual token-length bottleneck:

- **Sparse Mixture-of-Experts (MoE) decouples total capacity from per-token compute:** When a single high-resolution document or video adds thousands of tokens, keeping active parameters small per token keeps prefill and decode tractable.
  - **Qwen3.8-Flash-Next** (Aug 2026): 125B total parameters, 6B active (512 experts, 10 active + 1 shared).
  - **DeepSeek-V4.1-Flash** (Sept 2026): 552B backbone parameters, 16B active per token during decode and 8B during prefill (384 routed experts, 6 active, 1 shared).
  - **Kimi K3 / K3.1** (July 2026): 2.8T total parameters, 104B active (896 routed experts, 16 active, 2 shared, 1M context).
- **Hybrid and sparse attention for 1M context makes long video and multi-document vision tractable:**
  - **Qwen3.8-Flash-Next:** $3:1$ Gated DeltaNet to full-attention layers, with the full layers replaced by Qwen Sparse Attention (QSA) during continued pretraining.
  - **DeepSeek-V4.1-Flash:** CSA2 sparse attention with cross-layer KV reuse plus an FP4 KV cache requiring only 890 bytes per token of global KV.
  - **MiniMax-M3** (June 2026): Blockwise top-k sparse attention cutting per-token attention compute by $28.4\times$ at 1M context.
- **Reasoning post-training pipelines:** SFT $\rightarrow$ RL $\rightarrow$ on-policy distillation from expert teachers—as seen in **DeepSeek-V4.1-Flash** (distilling from > 40 teachers) and **Nemotron 3.5 Super / Lightning** (Oct 2026: SFT $\rightarrow$ RLVR $\rightarrow$ expert teachers $\rightarrow$ MOPD).

KEY IDEA

Modern modular VLMs combine native dynamic resolution, 2×2 spatial merging, Interleaved MRoPE, explicit timestamps, DeepStack multi-layer fusion, and square-root-normalized loss with low-active-parameter MoE and sparse 1M-context backbones to scale cleanly across high-resolution documents and long videos.

Yet every single model we have examined from LLaVA through Qwen3-VL shares one hard limitation: they are **understanding-only**. Continuous vision encoders inject images into the input stream, but the Transformer's output head predicts solely over a discrete text vocabulary—the model cannot emit an image or audio waveform directly. Answering our second guiding question requires stepping into multimodal generation.

✎ Check Your Understanding

1. **Quick computation:** An image of size 448×280 pixels enters Qwen2-VL's ViT with 14×14 patches and a 2×2 spatial merge. How many merged visual tokens (excluding boundary tokens) are sent to the LLM?

   Answer: The patch grid is $(448/14) \times (280/14) = 32 \times 20 = 640$ patches; after 2×2 spatial merging, the LLM receives $16 \times 10 = 160$ merged visual tokens.

2. **Yes or No:** In MRoPE, does setting $t = h = w$ for text tokens require retraining the LLM's text positional embeddings from scratch? Justify your answer.

   Answer: **No.** Setting $t = h = w = m$ applies the exact same scalar index m to every rotary channel pair, which reduces MRoPE identically to standard 1D RoPE and preserves full compatibility with pretrained text weights.

Section 6: Early-Fusion Discrete Multimodal Models — VQ-VAE and Chameleon

Recall the two routes to multimodal generation introduced in Section 1:

1. **External generative decoder:** Keep continuous encoders for understanding, and attach a separate diffusion decoder conditioned on Transformer hidden states to synthesize pixels.
2. **Early-fusion discrete tokens:** Quantize images into discrete token IDs drawn from a finite codebook so a *single* autoregressive Transformer reads and generates both text and images with the exact same next-token prediction objective.

DEFINITION

Early-fusion discrete modeling: A multimodal paradigm in which images and text are both converted into discrete token IDs before entering the model, allowing a single autoregressive Transformer to perform understanding, generation, and interleaved reasoning over one shared vocabulary.

Chameleon (2024) is the landmark realization of early-fusion discrete modeling. To turn a continuous 512×512 RGB image into discrete integer IDs that can also be decoded back into pixels, Chameleon uses a **VQ-VAE** (Vector Quantized Variational Autoencoder) image tokenizer.

DEFINITION

Vector Quantized Variational Autoencoder (VQ-VAE): An autoencoder trained via reconstruction loss that compresses an input image into a spatial grid of continuous latent vectors and snaps each latent vector to its nearest entry in a learned finite codebook, yielding discrete token IDs that a paired decoder maps back to pixels.

Trace the five-stage VQ-VAE tokenization and reconstruction loop from left to right:

1. **Image x (512×512):** The raw RGB input image.
2. **Encoder:** Maps x down to a lower-resolution spatial grid of continuous latent vectors $z_e(x)$.
3. **Vector Quantization (Codebook lookup):** Each continuous latent vector z_e is matched to its closest vector e_j inside a learned **Codebook** of size $8{,}192$:

$$k = \arg\min_j \|z_e - e_j\|_2$$

   The integer index $k \in \{0, 1, \ldots, 8191\}$ becomes the discrete visual token ID, and the corresponding codebook vector $z_q = e_k$ replaces z_e. In Chameleon, each 512×512 image becomes $1{,}024$ **discrete token IDs** (a 32×32 spatial grid).
4. **Decoder:** Reconstructs an image $\hat{x}$ from the grid of quantized vectors z_q.
5. **End-to-end reconstruction feedback:** A dashed feedback path from **Recon.** $\hat{x}$ back to **Image** x shows that the encoder, codebook, and decoder are trained end-to-end by minimizing reconstruction error $\min \|x - \hat{x}\|$.

Breaking down the quantization rule $k = \arg\min_j \|z_e - e_j\|_2$ term by term: $z_e \in \mathbb{R}^{d_z}$ is the continuous feature vector emitted by the encoder at one spatial grid cell; $e_j \in \mathbb{R}^{d_z}$ is the j-th prototype vector in the codebook ($j = 0, \ldots, K-1$, with $K = 8{,}192$ in Chameleon); $\|z_e - e_j\|_2$ is the Euclidean distance between the continuous latent and codebook entry j; and $\arg\min_j$ picks the single index k minimizing that distance. Conceptually, vector quantization partitions continuous latent space into K Voronoi cells and replaces every point inside a cell by that cell's integer ID k.

> **WORKED EXAMPLE**
> Trace vector quantization on a single spatial latent vector $z_e = (0.3, 1.4)$ with a 3-entry 2D codebook $e_0 = (1.0, 0.0)$, $e_1 = (0.0, 2.0)$, and $e_2 = (-1.0, -1.0)$, and connect the result to Chameleon's spatial budget:
>
> 1. **Compute Euclidean distances $\|z_e - e_j\|_2$:**
>   - $j = 0$: $\sqrt{(0.3 - 1.0)^2 + (1.4 - 0.0)^2} = \sqrt{0.49 + 1.96} = \sqrt{2.45} \approx 1.565$
>   - $j = 1$: $\sqrt{(0.3 - 0.0)^2 + (1.4 - 2.0)^2} = \sqrt{0.09 + 0.36} = \sqrt{0.45} \approx 0.671$
>   - $j = 2$: $\sqrt{(0.3 + 1.0)^2 + (1.4 + 1.0)^2} = \sqrt{1.69 + 5.76} = \sqrt{7.45} \approx 2.729$
> 2. **Nearest-codebook assignment:** The minimum distance occurs at $k = 1$, so the tokenizer emits token ID `1` and passes $z_q = e_1 = (0.0, 2.0)$ to the decoder.
> 3. **Quantization error:** The residual difference $\|z_e - z_q\|_2 \approx 0.671$ is permanently discarded before the Transformer or decoder ever sees the token.
> 4. **Chameleon's information budget:** Encoding a 512×512 image into $1{,}024 = 32 \times 32$ tokens means each discrete token summarizes an entire 16×16 pixel patch using only $\log_2(8{,}192) = 13$ bits.
>
> Whenever a 16×16 patch contains tiny printed letters or sharp fine-grained strokes, snapping z_e to the nearest of $8{,}192$ codebook prototypes discards the subtle residual needed to keep small text crisp.

Chameleon trains a new joint BPE tokenizer so text tokens and the $8{,}192$ VQ-VAE image token IDs share a single discrete vocabulary. Picture an interleaved sequence flowing into a **Single Autoregressive Transformer**:

`[Draw] [a] [cat:] [<img>] [v_4102] [...] [v_0819] [</img>] [Here] [is...]`

Every token—whether text (`Draw`, `a`, `cat:`), image delimiters (`<img>`, `</img>`), or quantized image patches (`v_4102`, …, `v_0819`)—passes through the exact same embedding table, Transformer layers, and next-token cross-entropy loss. To emit an image, the Transformer simply generates `<img>`, autoregressively samples $1{,}024$ image token IDs, generates `</img>`, and hands the $1{,}024$ IDs to the VQ-VAE decoder to synthesize pixels.

Chameleon trained this unified Transformer in two stages:

- **Stage 1 (80% of training):** Large-scale unsupervised mixture of 2.9T **text-only tokens**, 1.5T **text/image pairs**, and 400B **interleaved text/image tokens**.
- **Stage 2 (20% of training):** 50% Stage 1 data mixed with 50% high-quality curated data.

The architectural elegance is undeniable: one model performs image analysis, image generation, and interleaved multimodal reasoning natively. Yet early-fusion discrete training immediately ran into a severe optimization failure: **training divergence at scale**.

> **COMMON MISCONCEPTION**
> You might think that once images have been mapped into discrete integer IDs just like BPE text tokens, you can train a standard text-LLM architecture on the mixed stream with standard recipes and expect smooth convergence. That breaks down at scale because text tokens and VQ image tokens have fundamentally different statistical entropies and activation scales, causing shared Transformer layers to become violently unstable and diverge.

Follow the instability chain from left to right:

1. **Modality mismatch: Text Tokens** (lower entropy) and **Image VQ Tokens** (higher entropy) enter the same Transformer blocks with markedly different entropy distributions and activation scales.
2. **Shared Layer Competition:** Inside shared attention and MLP layers, the two modalities compete to increase vector norms in the residual stream, driving **uncontrolled activation norm growth** and **softmax logit drift**.
3. **Training Divergence:** Unbounded query and key norms cause attention logits to explode, triggering **attention entropy collapse** (attention weights snapping onto single positions) and catastrophic **loss spikes at scale**.

Chameleon stabilized early-fusion training across trillions of mixed-modal tokens using two architectural and objective fixes: **QK-Norm** and **z-loss regularization**.

1. QK-Norm

> **DEFINITION**
> **QK-Norm:** Applying layer normalization (or RMS normalization) directly to the query (Q) and key (K) vectors inside every attention block prior to computing dot products, bounding attention logit magnitudes against uncontrolled activation norm growth.

Inside every attention head, QK-Norm modifies scaled dot-product attention to:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{\text{Norm}(Q)\,\text{Norm}(K)^\top}{\sqrt{d_k}}\right)V$$

Decomposing this equation hierarchically:

1. **Inner normalization ($\text{Norm}(Q)$ and $\text{Norm}(K)$):** Before any dot product is taken, every query row and key row is normalized to a controlled scale, stripping away arbitrary norm growth accumulated in the residual stream.
2. **Bounded similarity scores ($\frac{\text{Norm}(Q)\,\text{Norm}(K)^\top}{\sqrt{d_k}}$):** Dividing the normalized inner products by $\sqrt{d_k}$ guarantees that pre-softmax attention logits stay within a safe numerical range regardless of modality competition.
3. **Attention weights and value aggregation ($\text{softmax}(\cdot)V$):** Because logits cannot blow up, softmax maintains healthy gradient flow instead of collapsing to degenerate one-hot spikes.

As a single conceptual unit, QK-Norm breaks the feedback loop between residual activation norm growth and attention entropy collapse.

2. z-Loss Regularization

Even with attention stabilized, the final output softmax over the shared vocabulary suffers from **logit drift**. Recall that softmax is strictly shift-invariant: adding any constant c to every logit ℓ_v leaves every output probability unchanged:

$$\text{softmax}(\ell)_u = \frac{e^{\ell_u}}{\sum_v e^{\ell_v}} = \frac{e^{\ell_u + c}}{\sum_v e^{\ell_v + c}} = \text{softmax}(\ell + c)_u$$

Because cross-entropy loss $\mathcal{L}_{\text{CE}}$ depends only on $\text{softmax}(\ell)$, it exerts zero restoring force against uniform drift c. During mixed-modal training, output logits drift far away from zero, causing numerical overflow and loss spikes.

DEFINITION

z-loss regularization: An auxiliary penalty $\lambda(\log Z)^2$ added to cross-entropy loss, where $Z = \sum_v \exp(\ell_v)$ is the softmax partition function, anchoring $\log Z$ near zero to stop shift-invariant output logits from drifting.

The regularized training objective is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CE}} + \lambda \left(\log Z\right)^2, \qquad Z = \sum_v \exp(\ell_v)$$

Breaking this equation apart term by term:

1. **Partition function ($Z = \sum_v \exp(\ell_v)$):** The denominator of the output softmax summed across all vocabulary entries v (both text and image IDs).
2. **Log-partition function ($\log Z$):** Shifts linearly with any constant logit offset: replacing $\ell_v \to \ell_v + c$ multiplies Z by e^c, shifting $\log Z \to \log Z + c$.
3. **Quadratic penalty ($\lambda \left(\log Z\right)^2$):** Penalizes deviations of $\log Z$ away from 0 symmetrically in either direction, weighted by a small hyperparameter $\lambda > 0$.
4. **Total objective ($\mathcal{L}_{\text{CE}} + \lambda \left(\log Z\right)^2$):** Cross-entropy determines the relative gaps between logits (preserving exact probabilities), while z-loss pins down their absolute baseline near zero.

WORKED EXAMPLE

Demonstrate concretely why both QK-Norm and z-loss are required by testing two numerical perturbations.

Part A — QK-Norm against activation norm growth: Suppose a single attention head has $d_k = 2$ and query/key vectors $q = (3, 4)$ and $k = (3, 4)$ (both of L_2 norm 5).

- Without QK-Norm: the scaled dot product is $\frac{q^\top k}{\sqrt{2}} = \frac{25}{1.414} \approx 17.68$. If cross-modal norm competition scales both vectors up by $10\times$ to $q' = (30, 40)$ and $k' = (30, 40)$, the attention logit explodes $100\times$ to $\frac{2{,}500}{1.414} \approx 1{,}768$, completely saturating softmax (entropy collapse).
- With L_2 QK-Norm: $\text{Norm}(q') = \text{Norm}(q) = (0.6, 0.8)$ and $\text{Norm}(k') = \text{Norm}(k) = (0.6, 0.8)$. The scaled dot product remains fixed at $\frac{1.0}{\sqrt{2}} \approx 0.707$ regardless of the $10\times$ norm surge.

Part B — z-loss against shift-invariant logit drift: Consider a 2-token vocabulary where token 1 is the ground truth and $\lambda = 10^{-4}$.

- **Centered logits $\ell = (2, 0)$:**
  - $Z = e^2 + e^0 \approx 7.389 + 1 = 8.389$, so $\log Z \approx 2.127$.
  - $p_1 = e^2 / 8.389 \approx 0.8808$, giving $\mathcal{L}_{\text{CE}} = -\log(0.8808) \approx 0.1269$.
  - z-loss penalty: $10^{-4} \times (2.127)^2 \approx 4.52 \times 10^{-4}$ (tiny—barely perturbs $\mathcal{L}_{\text{CE}}$).
- **Drifted logits $\ell' = (102, 100)$ ($c = +100$):**
  - $p_1 = \frac{e^{102}}{e^{102} + e^{100}} = \frac{e^2}{e^2 + 1} \approx 0.8808$, so $\mathcal{L}_{\text{CE}} \approx 0.1269$ is **completely blind** to the $+100$ drift.

- Yet $\log Z' = \log\big(e^{100}(e^2+1)\big) = 100 + 2.127 = 102.127$.
- z-loss penalty jumps to $10^{-4} \times (102.127)^2 \approx 1.043$—more than $8\times$ larger than $\mathcal{L}_{\text{CE}}$ itself—creating a strong restoring gradient that pulls the logits straight back toward zero.

KEY IDEA

VQ-VAE quantization lets an early-fusion Transformer like Chameleon read and generate images natively with pure next-token cross-entropy, but mixing low-entropy text and high-entropy visual tokens requires QK-Norm and z-loss to prevent activation norm growth, attention entropy collapse, and logit drift.

Chameleon proved that unified discrete autoregression can be stabilized across trillions of mixed-modal tokens. Why, then, have most practical omni systems *not* abandoned continuous vision encoders in favor of pure VQ-VAE tokenization? Answering that question reveals the central design trade-off of multimodal modeling.

✎ Check Your Understanding

1. **Fill in the blank:** In Chameleon's early-fusion stabilization, _____ bounds pre-softmax attention logits against activation norm growth, while _____ penalizes $(\log Z)^2$ to stop shift-invariant output logits from drifting.

   Answer: **QK-Norm**; **z-loss regularization**.

2. **Quick computation:** If every output logit ℓ_v shifts by $c = +50$ while keeping all relative differences identical, how much does $\mathcal{L}_{\text{CE}}$ change, and how much does $\log Z$ change?

   Answer: $\mathcal{L}_{\text{CE}}$ changes by 0 because softmax is shift-invariant ($\text{softmax}(\ell + 50) = \text{softmax}(\ell)$), while $\log Z$ increases by exactly $+50$ because Z is multiplied by e^{50}.

Section 7: The Comprehension-vs.-Generation Tension and Practical Omni Synthesis

Comparing continuous vision encoders (CLIP, SigLIP, SigLIP-2) head-to-head against discrete visual tokenizers (VQ-VAE / Chameleon) exposes a sharp asymmetry across every engineering dimension:

Property	Continuous Encoders (CLIP / SigLIP / SigLIP-2)	Discrete Tokenizers (VQ-VAE / Chameleon)
Token type	Continuous ViT patch vectors	Discrete codebook IDs (e.g., $8{,}192$)
Fine-grained detail & OCR	**High** — preserves subtle visual detail	**Degraded** — discretization loses fine detail (e.g., small text / OCR)
Image generation	Not native (needs external decoder)	**Native** — single autoregressive LM head
Training stability	Mature staged alignment	Harder — entropy mismatch needs QK-Norm + z-loss
Architectural simplicity	Requires encoder + adapter + decoder	**Elegant** — pure next-token modeling

This comparison is not merely a temporary engineering gap; it reflects a **fundamental tension between comprehension and generation**.

Picture two opposing requirements at the top of a convergence diagram:

- **Understanding (left):** Wants **high-level semantics** that are *invariant to pixel noise*—recognizing that a cat is a cat regardless of exact fur texture, lighting speckles, or camera grain, while preserving dense continuous capacity for reading fine text.
- **Generation (right):** Wants **fine-grained details**—exact textures, sharp edges, shading, and pixel layout—so synthesized images look photorealistic rather than blurry.

Contrastive encoders (CLIP, SigLIP, SigLIP-2) intentionally discard irrelevant pixel-level variations to isolate semantics. Reconstruction tokenizers (VQ-VAE) do the opposite: because they are trained to minimize pixel reconstruction error $\|x - \hat{x}\|$, they spend their finite codebook capacity on local textures and pixel fidelity, resulting in weaker high-level semantics and degraded OCR when fine text falls below the quantization grain.

COMMON MISCONCEPTION

You might think that unifying reading and writing into a single discrete VQ-VAE tokenizer is strictly superior to separating input encoders from output decoders, and must therefore outperform hybrid designs on every task. That breaks down because comprehension and generation place directly opposing demands on the same bottleneck: forcing one discrete codebook to simultaneously provide pixel-level texture fidelity and noise-invariant high-level semantics compromises both, especially fine-grained visual comprehension and OCR.

WORKED EXAMPLE

Trace this representational trade-off on a photograph of a rain-streaked street sign mounted in front of a textured brick wall:

1. **Continuous contrastive encoder (SigLIP-2):** Trained on OCR-enriched image-text alignment, its continuous patch embeddings ignore nuisance raindrop reflections and brick grain while allocating smooth, high-dimensional coordinates to the exact letter shapes on the sign and the high-level scene semantics. Reading the sign succeeds accurately, but those semantic vectors do not store enough low-level texture detail to reconstruct every brick and raindrop back into a sharp photograph.
2. **Discrete VQ-VAE tokenizer (Chameleon, $8{,}192$-entry codebook, 16×16 pixel patches):** Trained to minimize pixel reconstruction loss $\|x - \hat{x}\|$, each 13-bit token must encode the exact colors, edges, raindrop highlights, and mortar lines inside its 16×16 patch. Because codebook capacity is spent reproducing pixel textures everywhere in the frame, small letters on the sign snap to an approximate prototype entry, blurring characters and degrading OCR comprehension—even though the tokenizer can decode plausible-looking bricks and rain from discrete token IDs.

Both arrows from **Understanding** and **Generation** therefore converge into the **Prevailing Omni Architecture** (the third paradigm introduced in Section 1):

$$\text{Continuous Encoders} \longrightarrow \text{Multimodal Transformer} \longrightarrow \text{Diffusion Decoders (alongside Text LM Head)}$$

This hybrid synthesis **decouples semantic reading from pixel synthesis**: continuous encoders (such as SigLIP-2) supply high-accuracy, OCR-preserving visual inputs to the shared long-context Multimodal Transformer, while a diffusion decoder conditioned on the Transformer's output states synthesizes fine-grained textures, edges, and layouts without forcing the input encoder through a lossy discrete bottleneck.

Pulling the entire lecture together, four practical engineering levers govern the quality and scalability of modern multimodal systems:

1. **Data curation and synthesis dominate quality:** OCR-enriched web filtering (WebLI's top 10% across 100 languages), LLM-synthesized visual conversations and Long-CoT reasoning data (LLaVA, LLaVA-OneVision, Qwen3-VL), and easy-to-hard training curricula drive larger gains than raw uncurated scale alone.
2. **Resolution and token budgeting:** Native dynamic resolution, 2×2 spatial merging, AnyRes tiling with bilinear pooling, and modality-aware frame budgets keep image and video sequence lengths manageable inside finite context windows—reinforced by late-2026 sparse MoE and 1M-context sparse-attention backbones.

3. **Cross-modal stability and loss weighting:** Square-root-normalized per-token loss balances low-density video sequences against dense text during modular VLM training, while QK-Norm and z-loss prevent mixed-modal divergence in early-fusion discrete architectures.
4. **Spatial-temporal alignment across depth:** Interleaved MRoPE, explicit video timestamp tokens, and DeepStack cross-layer feature injection preserve 3D space-time geometry and fine-grained low-level cues all the way through deep language models.

KEY IDEA

Comprehension demands noise-invariant high-level semantics while generation demands fine-grained pixel reconstruction; the prevailing omni architecture resolves this tension by pairing continuous encoders with a shared long-context multimodal Transformer and diffusion decoders, governed in practice by data curation, token budgeting, cross-modal stability, and 3D spatial-temporal alignment.

✎ Check Your Understanding

1. **Yes or No:** In the prevailing hybrid omni architecture (continuous encoders + Multimodal Transformer + diffusion decoders), are input images discretized into a finite VQ-VAE codebook before entering the Transformer? Justify your answer.

   Answer: No. Input images pass through continuous vision encoders (such as SigLIP / SigLIP-2) to preserve high-level semantics and fine-grained OCR detail, while pixel synthesis is offloaded to a diffusion decoder at the output.

2. **What would happen if…** you replaced the continuous vision encoder in a document-OCR VLM with an $8{,}192$-code VQ-VAE tokenizer while keeping the rest of the pipeline identical?

   Answer: OCR and fine-grained comprehension would degrade because VQ-VAE discretization snaps each patch's continuous latent vector to a finite codebook prototype optimized for pixel reconstruction, discarding subtle character details.