Vision-Language Models

Lecture 18 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Turn an image into tokens (**ViT** patch embedding).
2. Explain **CLIP** contrastive training over image-text pairs.
3. Use CLIP for **zero-shot classification**.
4. Describe **LLaVA's** linear-projection bridge to an LLM.
5. Contrast **native vs bolt-on multimodal** architectures.
6. Diagnose **VLM hallucination** (language-prior override).

Where we are

- **CNNs** (L7–L9) · vision-specific inductive bias.
- **Transformers** (L13–L14) · originally for text.
- **Self-supervised** (L17) · contrastive and masked pretraining.

Today: what if one model handled **both** text AND vision?

> REFERENCE
>
> Today maps to **Prince Ch 12 §12.5** (ViT) + the CLIP, LLaVA, Flamingo papers. This is where the full "multimodal" LLM came from.

Four questions:

1. How do Transformers process images (no convolutions)?
2. What did **CLIP** unlock?
3. How does **LLaVA** give an LLM eyes?
4. What's the 2026 multimodal state?

PART 1

Vision Transformer (ViT)

Apply Transformer to images — no convolutions

The 2020 bet

> REFERENCE
>
> Dosovitskiy et al. 2020 · *"An Image is Worth 16×16 Words"* — split image into patches, treat them as tokens, apply vanilla Transformer. Drop convolutions entirely.

Controversial at the time — CNNs had reigned for 8 years. The bet: if you have enough data and compute, the right architecture is the one with the fewest inductive biases.

It worked. ViT-Huge pretrained on 300M images beat CNN SOTA on ImageNet by 2021.

Why throw away CNNs · the bet of 2020

KEY IDEA

CNNs have vision **baked in** · they are forced to look at local pixels first, then build up to objects.

The 2020 bet · *what if we hand a generic Transformer the whole image at once and let it figure out what's important?*

The wager · with enough data, a model that has **fewer prior assumptions** but **more capacity** will outlearn a hand-engineered architecture. By 2021 (ViT-Huge on 300M images) the bet paid off · ViT matched ResNet on ImageNet.

How can a Transformer "read" an image?

INTUITION

Analogy · describing a photo over the phone. You don't list every pixel. You break it into chunks: *"top-left, blue sky · below that, a green tree..."* We teach the model to do the same · chop the image into a grid of **patches** ("image words").

From pixels to tokens · with a tiny example

A 4×4 grayscale image, 2×2 patches. We get $(4/2)^2 = 4$ patches.

$$\text{Image} = \begin{pmatrix} 10 & 20 & 150 & 160 \\ 30 & 40 & 170 & 180 \\ 190 & 200 & 5 & 15 \\ 210 & 220 & 25 & 35 \end{pmatrix}$$

Step 1 · patchify. Patch 1 (top-left) = $\begin{pmatrix} 10 & 20 \\ 30 & 40 \end{pmatrix}$.

Step 2 · flatten. Patch 1 → $[10, 20, 30, 40]$.

Step 3 · linear-project. Multiply by learned W (size 4×3 for output dim 3) → 3-d embedding.

Step 4 · prepend `[CLS]`, add position embeddings. Sequence: `[CLS, p1, p2, p3, p4]` + `[pe_0, pe_1, …]`.

For real 224×224 RGB with 16×16 patches:

- Each patch · $16 \cdot 16 \cdot 3 = 768$ numbers.
- Project to $d = 768$. Patches per image · $(224/16)^2 = 196$.
- Final sequence · $1 + 196 = 197$ tokens. Standard Transformer from here.

The patchify step is conceptually a strided conv with kernel = stride = patch size.

Worked numeric · patch embedding

Tiny example. Flattened patch · $x = [10, 20, 30, 40]$.

Learned W (size 4×3):

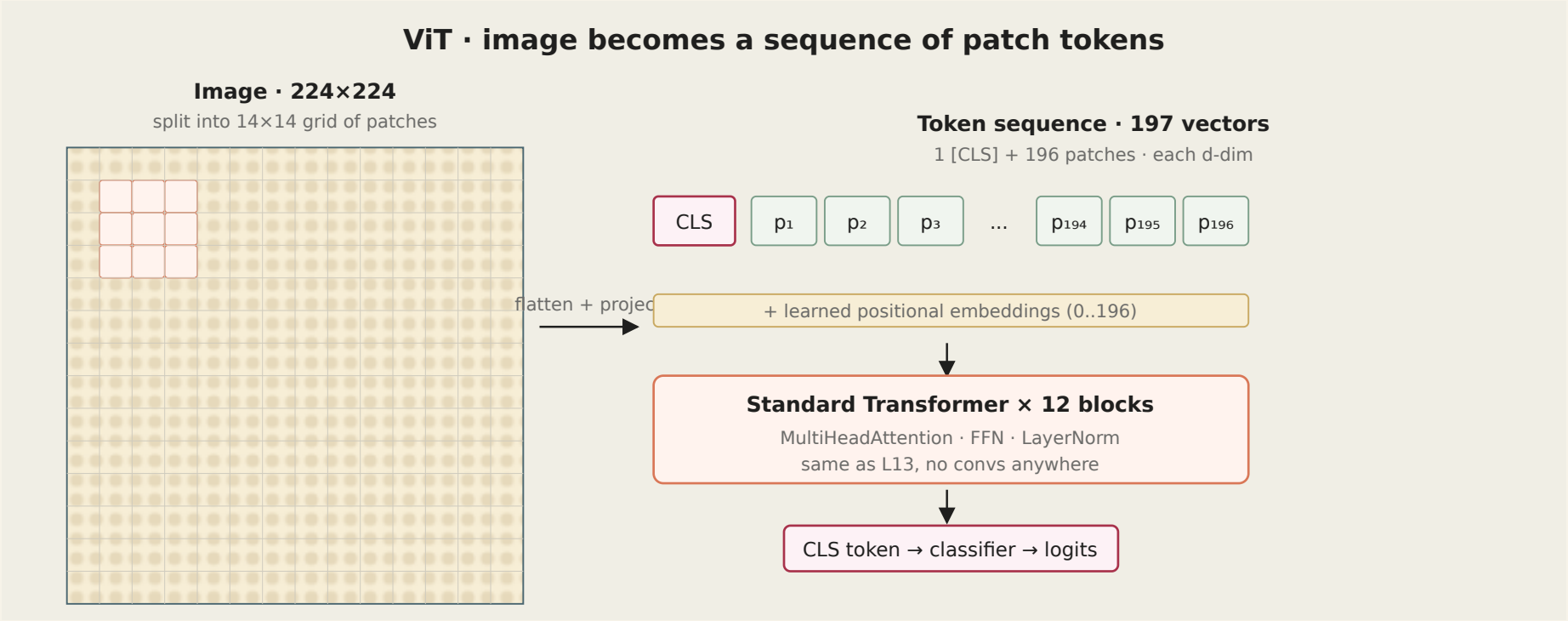
$$W = \begin{pmatrix} 0.1 & 0 & -0.2 \\ 0 & 0.5 & 0 \\ -0.1 & 0 & 0.3 \\ 0.2 & -0.1 & 0 \end{pmatrix}$$

Compute. $\text{embedding} = xW$.

- Component 1 · $10(0.1) + 20(0) + 30(-0.1) + 40(0.2) = 1 - 3 + 8 = 6.0$
- Component 2 · $10(0) + 20(0.5) + 30(0) + 40(-0.1) = 10 - 4 = 6.0$
- Component 3 · $10(-0.2) + 20(0) + 30(0.3) + 40(0) = -2 + 9 = 7.0$

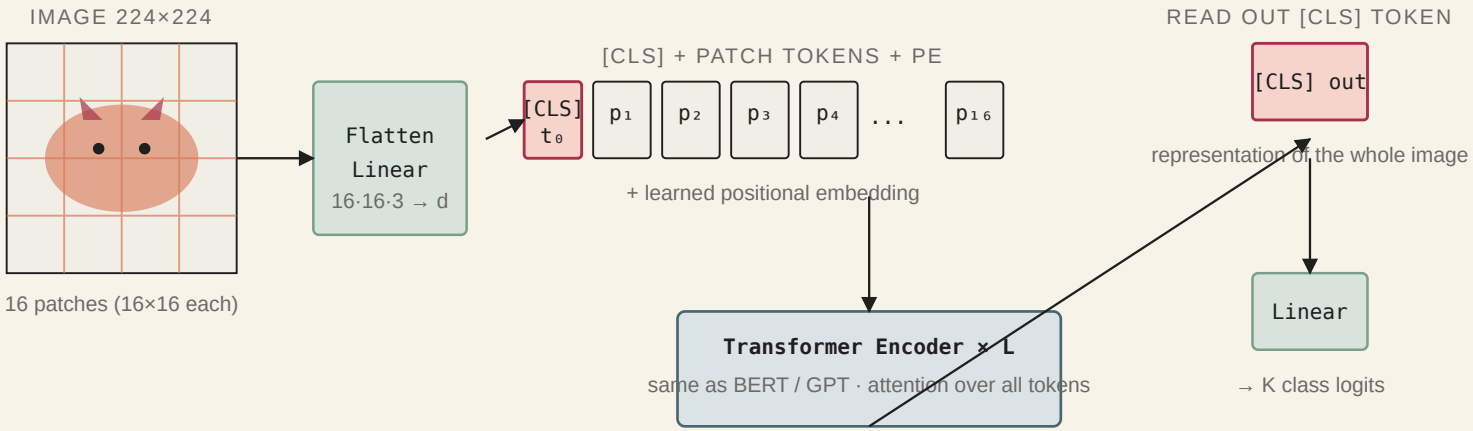
  $\text{embedding} = [\mathbf{6.0,\ 6.0,\ 7.0}]$ — the first "token" the Transformer sees, representing the top-left of the image.

From image to sequence · picture



How ViT works

Vision Transformer (ViT) — treat image patches as tokens, apply Transformer



KEY INSIGHT

No convolutions at all. ViT works only at **scale** — needs 300M+ images of pretraining (JFT-300M) to beat ResNet. With self-supervised pretraining (DINO, MAE), smaller data works tc

ViT vs CNN · inductive biases

	CNN	VIT
Receptive field growth	local → global slowly	global from layer 1
Weight sharing	spatial	none (each position has own attention)
Translation equivariance	baked in	learned (if at all)
Data efficiency (small data)	strong	weak
Data efficiency (large data)	plateaus	keeps improving

KEY IDEA

CNNs encode vision priors; ViTs learn them. With small data, priors win. With massive data (300M+), learning wins.

ViT variants you'll encounter

MODEL	PATCH	PARAMS	NOTES
ViT-B/16	16×16	86M	"Base" — most common
ViT-L/14	14×14	307M	Used by CLIP
ViT-H/14	14×14	632M	SOTA around 2021
ViT-g (DINOv2)	14×14	1.1B	2023 general-purpose vision

Swin Transformer (Liu et al. 2021) adds hierarchical windowing — bridges ViT and CNN. Popular in practice.

ViT in PyTorch · 30 lines

```
class ViT(nn.Module):
    def __init__(self, image_size=224, patch=16, dim=768, depth=12, heads=12, classes=1000):
        super().__init__()
        self.patch_embed = nn.Conv2d(3, dim, kernel_size=patch, stride=patch)   # patchify
        n_patches = (image_size // patch) ** 2
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.pos_embed = nn.Parameter(torch.randn(1, n_patches + 1, dim))
        self.blocks = nn.ModuleList([TransformerBlock(dim, heads) for _ in range(depth)])
        self.norm = nn.LayerNorm(dim)
        self.head = nn.Linear(dim, classes)

    def forward(self, x):
        x = self.patch_embed(x).flatten(2).transpose(1, 2)      # (B, N, D)
        cls = self.cls_token.expand(x.size(0), -1, -1)
        x = torch.cat([cls, x], dim=1) + self.pos_embed
        for b in self.blocks: x = b(x)
        return self.head(self.norm(x[:, 0]))                   # CLS → class logits
```

Literally a Transformer from L13 with a patch-embed preamble. No vision-specific ops.

PART 2

CLIP · contrastive image-text pretraining

The paper that launched zero-shot vision

CLIP · training as contrast matrix

CLIP · batch of N image-text pairs · N positives on the diagonal, N² − N negatives

Image encoder

ViT-L/14

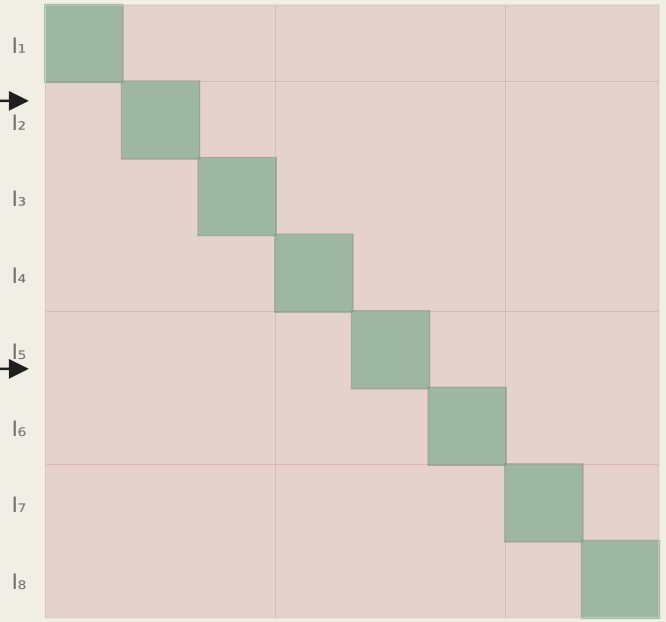
imgs → I₁..I_N

Text encoder

12-layer Transformer

captions → T₁..T_N

Similarity matrix I·Tᵀ (8×8 example · scale is N=32k)



Loss

Symmetric cross-entropy

per-row softmax pulls diagonal up, pushes row-negatives down.

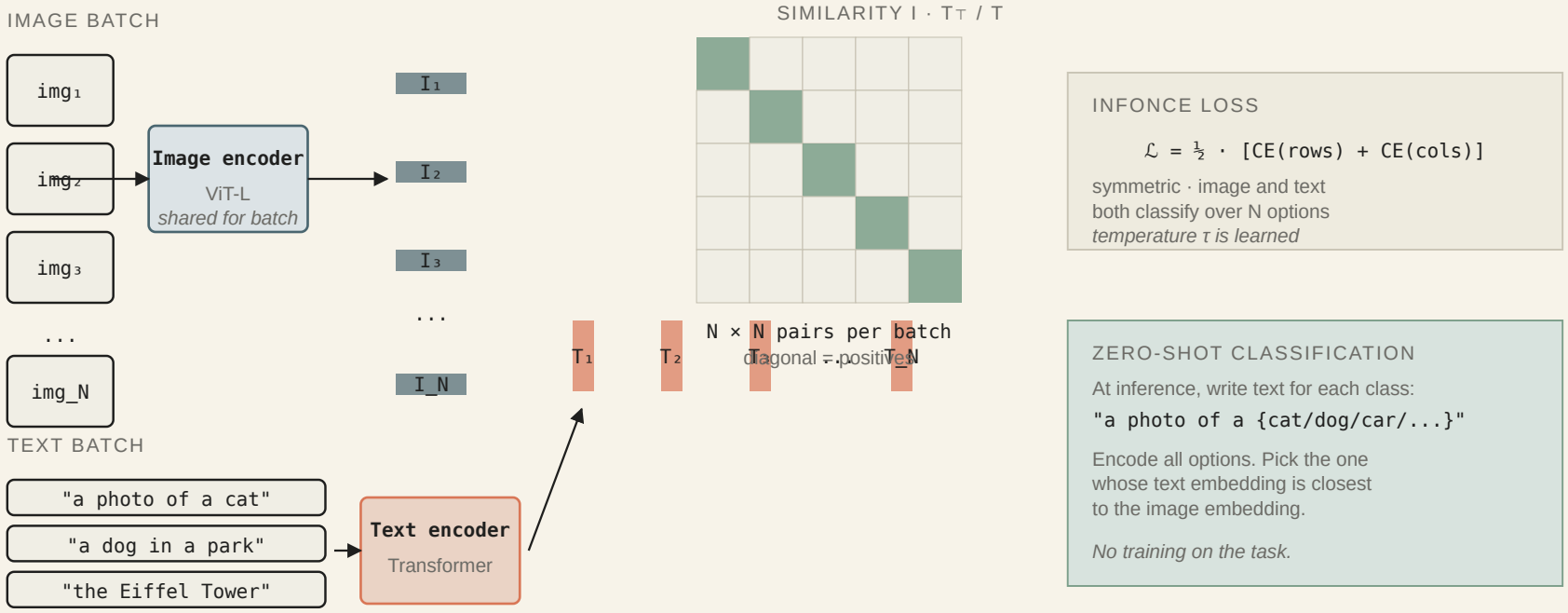
Scale (2021)

400M image-text pairs
Batch size · 32,768
256 A100 GPUs · 12 days

→ 1 positive vs 32,767 negatives
→ extremely rich training signal

CLIP · dual encoder

CLIP — dual encoder · align image and text embeddings via contrastive loss



CLIP · the core idea

Train on (image, caption) pairs scraped from the web. Push matching pairs close in embedding space, push mismatched pairs far apart.

KEY IDEA

Build a **shared space** where images and captions live together. Once it exists, you can:

- **Retrieve images** from a text query (nearest caption embedding).
- **Classify zero-shot** · pick the closest *caption template* to the image.
- **Guide generation** · diffusion models condition on CLIP text features.

The training objective is InfoNCE (L17), extended across two modalities instead of two augmentations.

CLIP · the matching-game derivation

Tiny batch · $N = 3$ image-text pairs $(I_1, T_1), (I_2, T_2), (I_3, T_3)$.

1. **Step 1 · embed.** Encode each image and text → unit vectors i_1, i_2, i_3 and t_1, t_2, t_3.
2. **Step 2 · similarity matrix.** $S_{ij} = i_i \cdot t_j$.

   $$S = \begin{pmatrix} i_1 \cdot t_1 & i_1 \cdot t_2 & i_1 \cdot t_3 \\ i_2 \cdot t_1 & i_2 \cdot t_2 & i_2 \cdot t_3 \\ i_3 \cdot t_1 & i_3 \cdot t_2 & i_3 \cdot t_3 \end{pmatrix}$$

   Diagonal = correct matches → maximize. Off-diagonal → minimize.
3. **Step 3 · scale by temperature.** $\text{logits} = S/\tau$.
4. **Step 4 · row-wise CE.** Each row of logits → softmax → CE against the diagonal label. (Image I_1 should match text T_1.)
5. **Step 5 · column-wise CE.** Same for columns (text T_1 should match image I_1).
6. **Final loss.** $\mathcal{L} = \frac{1}{2}(\text{CE}_{\text{rows}} + \text{CE}_{\text{cols}})$.

- **Image encoder** · ViT-L/14 (or ResNet-50 for small).
- **Text encoder** · 12-layer Transformer.
- **Data** · 400M image-text pairs from the web. **Batch size** 32,768.

Worked numeric · CLIP loss for one row

$N = 2$. Normalized embeddings:

- $i_1 = [0.8, 0.6]$, $t_1 = [0.7, 0.7]$ (good match)
- $i_2 = [-0.9, 0.44]$, $t_2 = [-0.8, 0.6]$ (good match)
- $\tau = 0.1$.

Similarity matrix.

- $i_1 \cdot t_1 = 0.56 + 0.42 = 0.98$ ✓
- $i_1 \cdot t_2 = -0.64 + 0.36 = -0.28$

$S_{\text{row } 1} = [0.98, \ -0.28]$

Logits row 1. $S_{\text{row } 1} / 0.1 = [9.8, \ -2.8]$.

Softmax. $\exp([9.8, -2.8]) \approx [18034, 0.061]$. Normalize → $[0.99997, 0.00003]$.

Loss row 1 (correct = index 0). $-\log 0.99997 \approx \mathbf{3 \times 10^{-5}}$ — very low.

If the off-diagonal were higher, the loss spikes immediately. CLIP just runs this over 32k images simultaneously.

CLIP's killer feature · zero-shot

```
import clip
model, preprocess = clip.load("ViT-L/14")

image = preprocess(pil_image).unsqueeze(0)
texts = ["a photo of a cat", "a photo of a dog", "a photo of a car"]
text_tokens = clip.tokenize(texts)

with torch.no_grad():
    img_emb  = model.encode_image(image)
    txt_embs = model.encode_text(text_tokens)

# Cosine similarity → pick the closest text
sims = (img_emb @ txt_embs.T).softmax(dim=-1)
# sims = [[0.89, 0.08, 0.03]]  → "cat"
```

No training on cats. CLIP has never seen an ImageNet label. Yet it beats ResNet-50 on zero-shot ImageNet classification.

Worked example · zero-shot a single image

Image · a photo of a tabby cat. CLIP image encoder produces a 768-dim vector (after L2-normalize · unit length).

DERIVATION

Build text prompts · `"a photo of a {cat / dog / car}"`. Run through CLIP text encoder.

CLASS	TEXT EMB	COS · IMG EMB	SOFTMAX
cat	`t_cat`	0.32	**0.89**
dog	`t_dog`	0.19	0.08
car	`t_car`	0.04	0.03

Pick `argmax` · "cat" wins with 89% confidence. **Total inference cost · 1 image-encoder call + 3 text-encoder calls.** No fine-tuning. Add 100 classes · still 100 text-encoder calls (a few seconds, total) and you're done.

Why CLIP mattered

1. **Universal image representation** — one encoder works on any domain.
2. **Prompt engineering for vision** — "a photo of X" vs "a sketch of X" vs "a satellite image of X" shifts behavior without retraining.
3. **Foundation for everything that came next** — Stable Diffusion uses CLIP's text encoder; DALL-E 2, Flamingo, LLaVA all build on CLIP features.

> IN PRACTICE
>
> CLIP is the **default general-purpose vision-language model** in 2026. For retrieval, search, zero-shot classification, content moderation, CLIP features are the first thing you try.

Prompt engineering · for vision

Zero-shot CLIP is sensitive to how you phrase the class label. Changing the prompt template affects accuracy by **~10% absolute on ImageNet.**

TEMPLATE	IMAGENET ZERO-SHOT
`"{label}"` (bare word)	63%
`"a photo of a {label}"`	68%
`"a photo of a {label}, a type of pet"` (for pets dataset)	72%

INTUITION

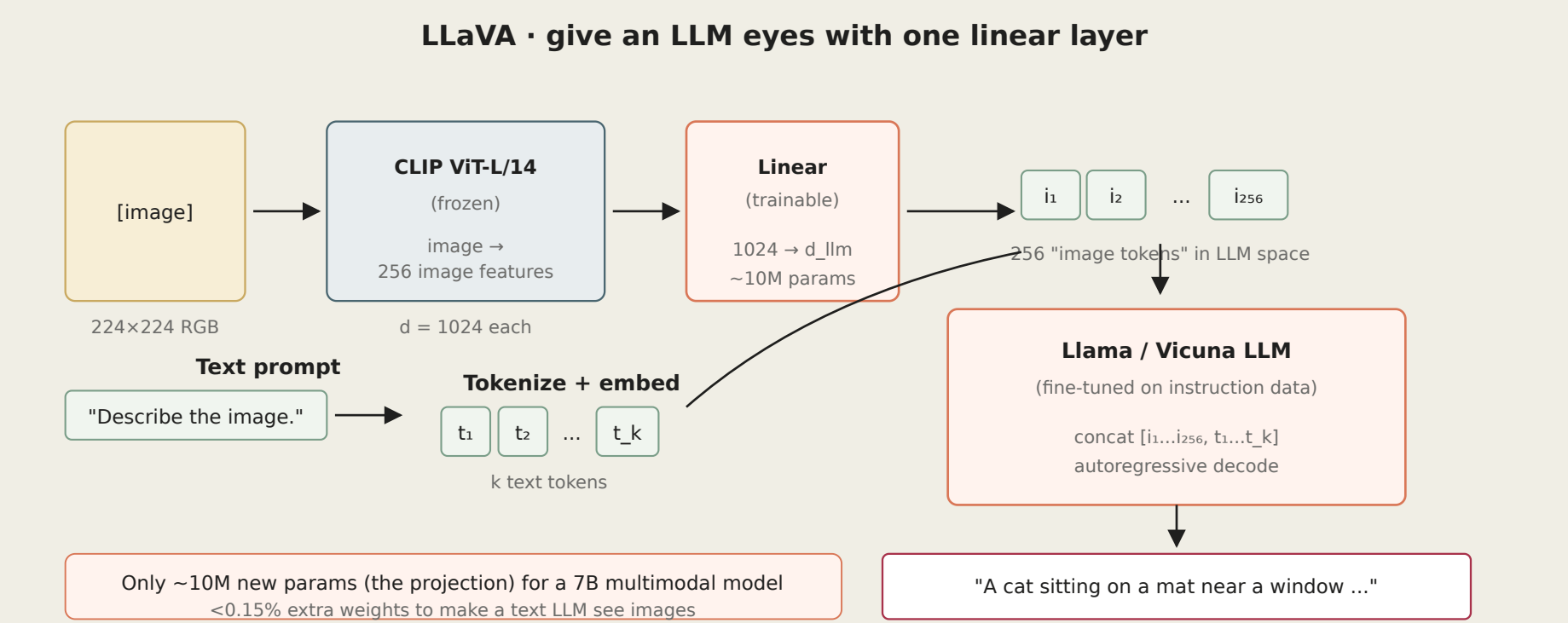
The model's "understanding" of a class is mediated by the caption distribution it was trained on. If most cat images were captioned "a photo of a cat", matching that template works best. Prompt-engineering for vision is *not* a trick — it's matching the training distribution.

PART 3

LLaVA · give an LLM eyes

Project image features into token space

LLaVA · full stack



LLaVA · the architecture

```
image → ViT-L/14 (CLIP encoder) → 256 image tokens (frozen)
                                          ↓
                                  Linear projection
                                          ↓
                                "Image tokens" in LLM space
                                          ↓
                          Llama 2 / Vicuna (autoregressive)
                                          ↓
                                    text response
```

DERIVATION

LLaVA recipe (Liu et al. 2023):

1. Pretrained CLIP ViT-L extracts 256 image features.
2. Simple MLP projects them into the LLM's token embedding space.
3. Concatenate `[image_tokens, text_tokens]` and feed to an LLM.
4. Fine-tune with instruction data: `(image, question, answer)` triples.

Surprisingly good. The LLM brings reasoning; CLIP brings vision understanding; the projection layer glues them.

LLaVA · the translator analogy

INTUITION

Two experts speak different languages.

- **CLIP** understands images (vector language $\mathbb{R}^{1024}$).
- **Llama** understands text (vector language $\mathbb{R}^{4096}$).

We need a **translator** · a linear map W that converts an image vector into something Llama can read. Both spaces already encode similar concepts (puppy near dog, etc.) — the translator only needs to rotate and scale.

LLaVA · the linear projection, with shapes

1. **CLIP output** · 256 patch embeddings, each 1024-d. Tensor `[256, 1024]`.
2. **LLM expects** · token embeddings of dim 4096.
3. **Bridge** · linear layer `W` of shape `[1024, 4096]` plus bias `[4096]`.

Per patch: `llm_token = patch @ W + b`. Shape check · `[1, 1024] @ [1024, 4096] → [1, 4096]` ✓.

Do this for all 256 patches → 256 vectors that "look like" tokens to the LLM. Prepend them to the user's text:

$$[\text{img}_1, \ldots, \text{img}_{256}, \text{text}_1, \text{text}_2, \ldots] \rightarrow \text{LLM}$$

Stage 1 · freeze CLIP and LLM, train only W, b on caption data.
Stage 2 · unfreeze the LLM, fine-tune on instruction data.

New params · ~10M in the projection. A 7B LLM becomes multimodal with < 0.15% extra weights.

Worked numeric · the projection

CLIP patch · $\text{patch} = [2.0, -1.0, 0.5]$ (3-d for clarity). LLM expects 4-d tokens.

$$W = \begin{pmatrix} 1 & 0 & 0.5 & 0.1 \\ 0 & 2 & 0 & -0.2 \\ 0.1 & -1 & 0 & 0 \end{pmatrix}, \quad b = [0, 0, 0.1, 0]$$

Compute $\text{patch}@W$:

- $2.0 \cdot 1 + (-1.0) \cdot 0 + 0.5 \cdot 0.1 = 2.05$
- $2.0 \cdot 0 + (-1.0) \cdot 2 + 0.5 \cdot (-1) = -2.5$
- $2.0 \cdot 0.5 + (-1.0) \cdot 0 + 0.5 \cdot 0 = 1.0$
- $2.0 \cdot 0.1 + (-1.0) \cdot (-0.2) + 0.5 \cdot 0 = 0.4$

$\text{llm\_token} = [2.05, -2.5, 1.0, 0.4] + [0, 0, 0.1, 0] = \mathbf{[2.05, -2.5, 1.1, 0.4]}$

The LLM treats this just like the embedding for the word "cat".

Flamingo · cross-attention bridge

Alayrac et al. 2022 · an alternative approach:

- Keep the LLM frozen.
- Insert **cross-attention** layers between LLM blocks that attend to image features.
- Use a **Perceiver Resampler** — a learned set of query tokens that distill image features.

Benefit: the LLM never "sees" images as tokens; it queries them via cross-attention when it needs to.

> INTUITION
>
> Both approaches work. LLaVA is simpler and became dominant in open-source; Flamingo-style crossattention persists in frontier labs (Anthropic's vision, Google's Gemini variants).

PART 4

Multimodal LLMs in 2026

The frontier

Native-multimodal vs bolt-on · adopt or raise?

INTUITION

You want a dog that understands verbal **and** hand commands.

- **Bolt-on (LLaVA)** · adopt a brilliant adult dog (LLM) that knows verbal commands. Hire a translator (projection) to whisper hand-signal meanings. Cheap, fast — but the dog's brain never natively saw signals.
- **Native (Gemini, GPT-4o)** · raise a puppy from birth using both. Brain processes them as fundamental, intertwined inputs. Deeper understanding — but you must pretrain from scratch on multimodal data.

Native vs bolt-on · trade-offs

Bolt-on (LLaVA, Flamingo)

- Vision tower stays "foreign" to the LLM.
- Often weaker on tight text-vision interaction.
- **Cheap** — only train the bridge.

Native (Gemini, GPT-4o)

- Unified tokenization across modalities.
- Stronger multi-modal reasoning.
- **Expensive** — pretrain from zero on interleaved data.

INTUITION

2026 frontier leans **native**. Bolt-on dominates open-source · only feasible approach when you can't pretrain a 100B+ model from scratch.

Cross-attention · Flamingo's design

Instead of converting the image into LLM-space tokens, Flamingo keeps vision features *separate* and injects them via **cross-attention layers inserted between LLM blocks**.

DERIVATION

- Visible to LLM · text tokens (normal self-attention).
- Every few LLM blocks · an added cross-attention layer that queries image features.
- Image features condensed via a **Perceiver Resampler** · learned query tokens distill arbitrary-resolution images into a fixed set.

INTUITION

Benefit · LLM never "sees" pixels as tokens; it *asks* for image info when it needs it. Drawback · more complex architecture, more params to train. LLaVA's simpler "concat tokens" won on ease; Flamingo-style persists in frontier labs.

2026 multimodal state

MODEL	WHAT IT SEES	WHAT IT DOES
GPT-4V / GPT-5	image, video	general reasoning + tool use
Claude 4 (Anthropic)	image, PDF, video frames	general reasoning + computer use
Gemini 2 Ultra (Google)	image, audio, video	natively multimodal from pretraining
LLaVA / Qwen-VL (open)	image	open-source equivalent of GPT-4V

Key trend: the input side is increasingly "anything", the output is still mostly text. True any-to-any (text → image → video → audio) is the 2026+ frontier.

Practical multimodal prompting

```
from anthropic import Anthropic
client = Anthropic()

response = client.messages.create(
    model="claude-4",
    max_tokens=1024,
    messages=[
        {
            "role": "user",
            "content": [
                {"type": "image", "source": {"type": "url", "url": "https://..."}},
                {"type": "text", "text": "What does this plot suggest?"}
            ]
        }
    ]
)
```

Three lines in; a paragraph of reasoning out. The API abstracts away all the CLIP + projection + LLM machinery.

VLM benchmarks · what we measure

DERIVATION

BENCHMARK	TESTS	EXAMPLE TASK
VQAv2	general visual QA	"what color is the car?"
GQA	compositional reasoning	"is the animal to the left of the tree brown?"
MMMU	multi-discipline expert	college-level chemistry diagram
DocVQA	document understanding	read text from scanned form
ChartQA	chart reading	"what was Q3 revenue?"
AI2D	diagram reasoning	"which step comes after X?"

Claude / GPT-4o / Gemini all push 85%+ on VQAv2 now. MMMU remains hard (50-60%) · genuine multi-modal reasoning is the frontier.

Hallucination · priors fighting evidence

KEY IDEA

The LLM has seen "a cup of coffee on a desk" **millions of times** in text. The vision encoder gives it a blurry image of a desk.

If the visual signal is weak, the language **prior** can override it · the model adds a coffee cup that isn't there.

This is a battle between **prior** (text statistics from training) and **evidence** (current visual input). When evidence is strong (clear photo), priors don't matter much. When evidence is weak, they take over and the model "fills in" things confidently.

Why VLMs hallucinate

Vision-language models sometimes describe things that *aren't in the image*:

> *Image · a cat on a mat.*
> *Prompt · "describe the room in detail."*
> *Output · "... next to a blue vase on the wooden table ..."*

WATCH OUT

Cause · the language model has a strong prior from text pretraining ("rooms have vases") that overrides weak visual signal. Fixes · stronger vision encoder, more VQA training data, CFG-style text-image alignment.

GPT-4o and Claude 3.5+ dramatically reduced hallucination via more curated training and RLHF specifically on vision tasks. Still not solved.

Emerging applications

Vision tasks, zero-shot

- Segment Anything prompts
- OCR without explicit training
- Diagram understanding
- Chart reading

Agentic loops

- **Computer use** — click buttons from screenshots
- **Robotic manipulation** — VLM → action tokens
- **Content moderation**
- **Code + diagram refactoring**

The agentic side (Claude computer use, GPT operator) is where multimodal is most valuable in 2026.

Summary · Lecture 18 — summary

- **ViT** · patches → tokens → Transformer. No convolutions. Scales with data; CNNs plateau.
- **CLIP** · dual encoder, contrastive on 400M image-text pairs · zero-shot vision via text prompts.
- **LLaVA** · CLIP features → linear projection → LLM token space. Simple, effective.
- **Flamingo** · cross-attention bridge with Perceiver Resampler. Alternate approach.
- **2026** · every frontier LLM is multimodal (GPT-5, Claude 4, Gemini 2). Output side still text-dominated.

Read before Lecture 19

Prince Ch 17 · Variational Autoencoders.

Next lecture

Autoencoders & VAEs — compression, latent spaces, the reparameterization trick, ELBO.

> NOTEBOOK
>
> **Notebook 18** · `18-clip-zero-shot.ipynb` — load pretrained CLIP; zero-shot classify custom images; sweep text prompts to see how wording changes the result.