ELV26 Tutorial 3: **Video Learning**

Ellis Brown
2026-02-10

Overview

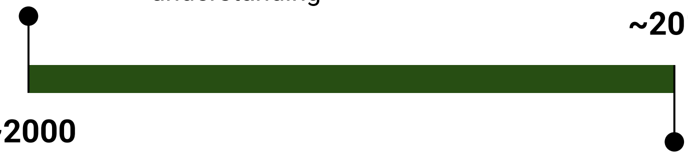
I. Context
II. Working w/ Video Data
III. Towards Video-LLMs
IV. Practical Training
V. Video-LLM failure modes

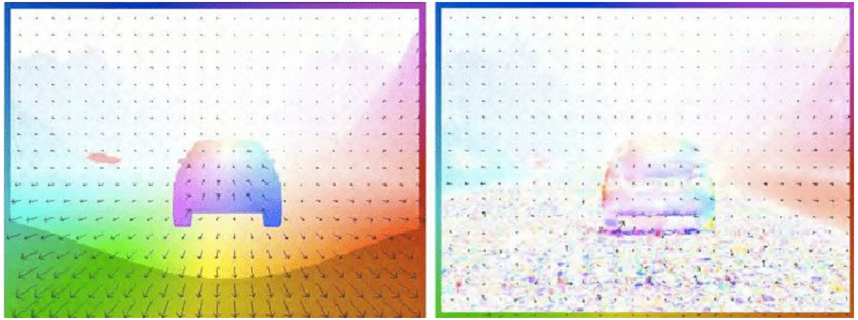
I. Context

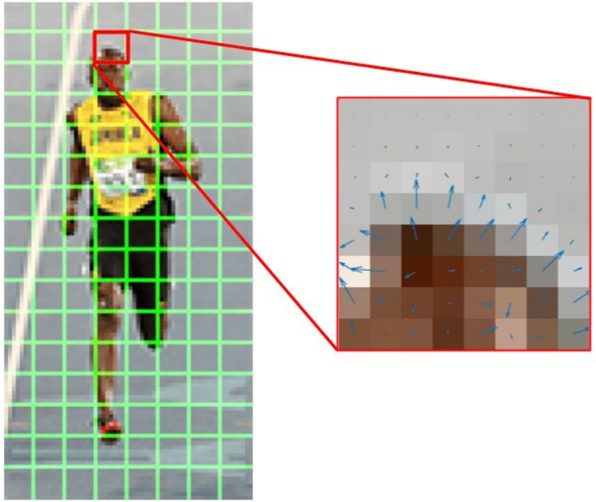
Birds-eye-view

Era 1 — Hand-Crafted Motion

- **Optical Flow**, hand-crafted features (eg, Dense Trajectories)
- Track pixel movements
- ❌ low-level, no semantic understanding







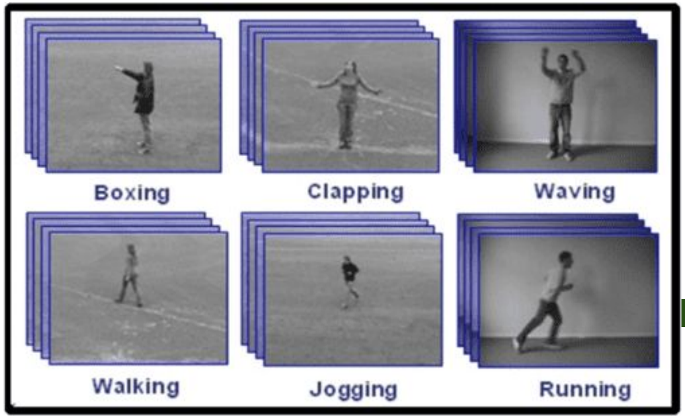
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

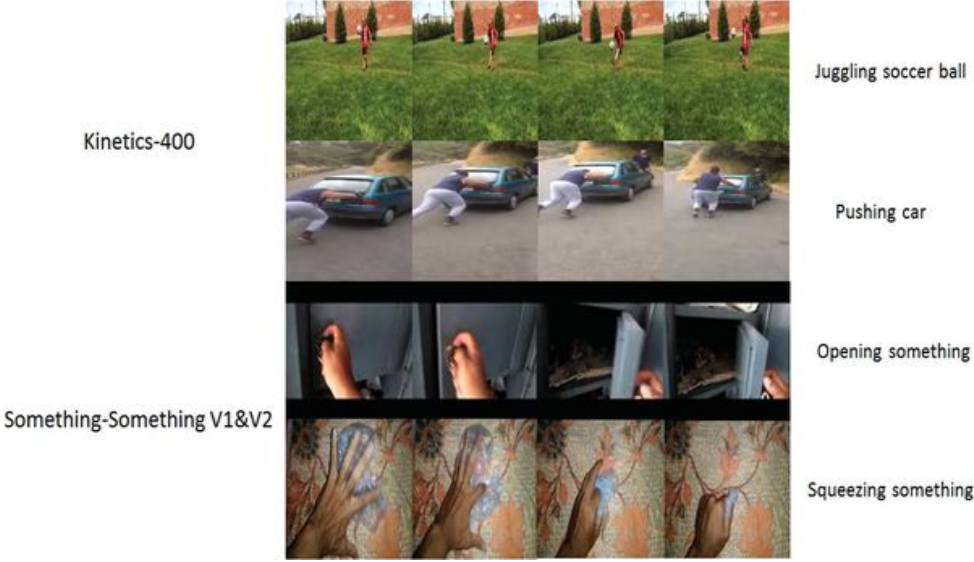
Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Birds-eye-view



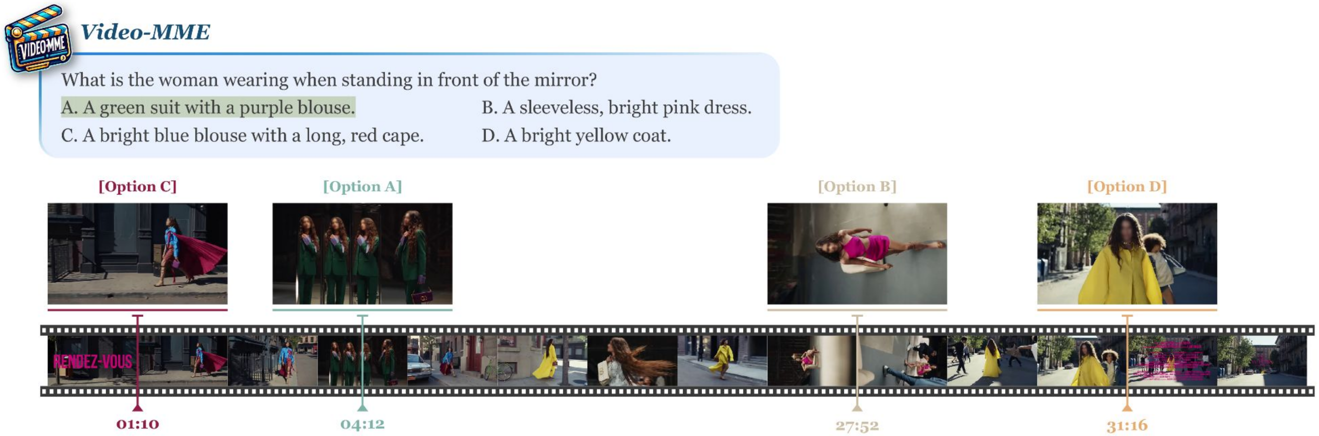


~2014

~2022

Era 2 — Deep Classification

- **3D** CNNs, Video-SSL
- Task: N-way classification (eg Kinetics-400 human action recognition)
- → shortcut: classify w/ one frame…



VSI-Bench

Era 3 — Generative *Understanding*

- Labels → *explanations*
- *Reasoning* about time, physics, intent
- Strong semantic understanding, *poor basic physical intuition*

~2022

Birds-eye-view

~2000

Era 1 — Hand-Crafted Motion

- **Optical Flow**, hand-crafted features (eg, Dense Trajectories)
- Track pixel movements
- ❌ low-level, no semantic understanding

~2014

Era 2 — Deep Classification

- **3D** CNNs, Video-SSL
- Task: N-way classification (eg Kinetics-400 human action recognition)
- → shortcut: classify w/ one frame…

~2022

Era 3 — Generative *Understanding*

- Labels → *explanations*
- *Reasoning* about time, physics, intent
- Strong semantic understanding, *poor basic physical intuition*

II. Working w/ Video Data

General deep learning process

Dataloading

1. Load raw data
2. Preprocess data
3. Gather data samples into batches

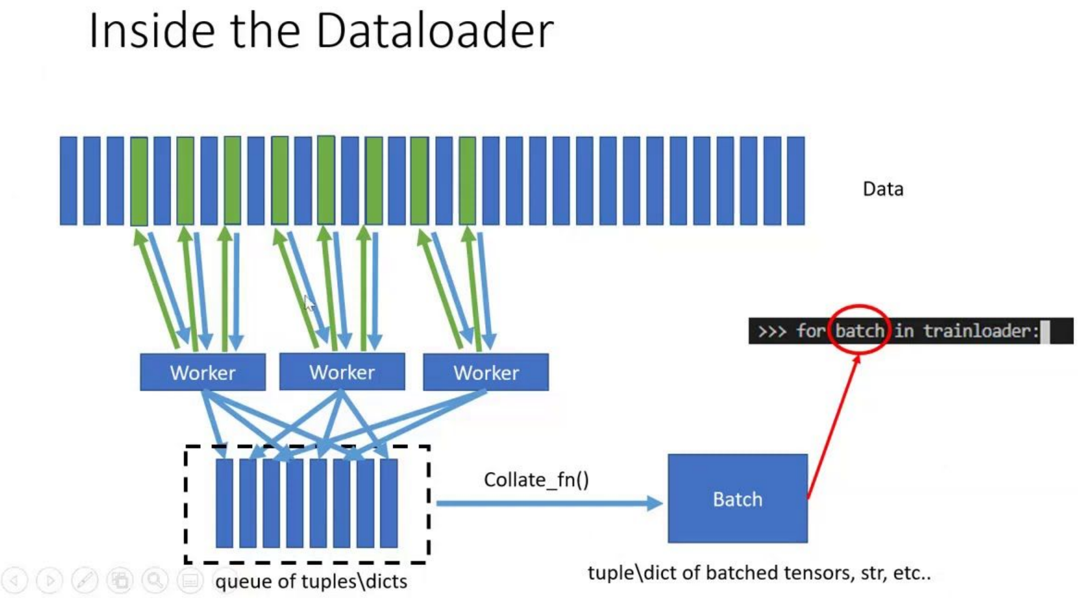
Model training

1. Load data batch onto GPU
2. Perform model forward pass and loss computation
3. Perform backpropagation to update model parameters

Dataloading

Data workers run on CPU cores

Runs asynchronously while model training is done on GPU



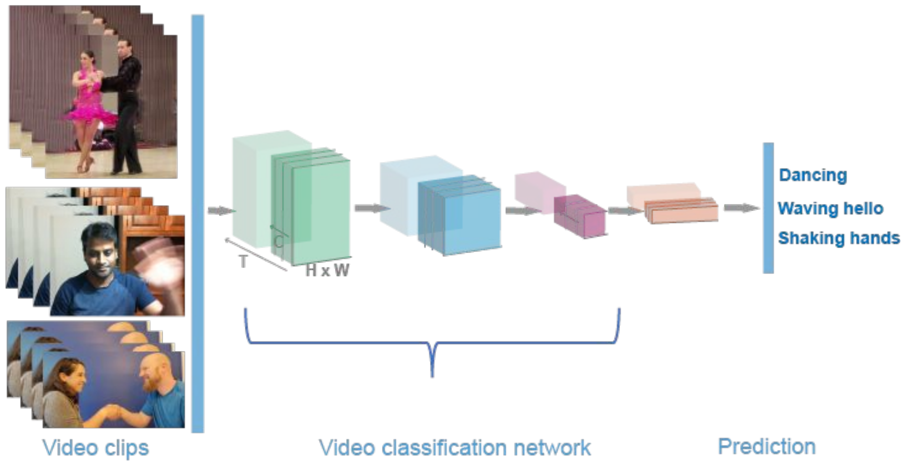
https://www.youtube.com/watch?v=Sj-gIb0QiRM

Videos

Sequences of image frames

Store key frames and “deltas” between key frames to achieve storage (lossy) compression

Enable learning temporal information, motion, object behavior, world models



Video datasets

Kinetics400: clips of human actions

Walking Tours: walking city tours

SomethingSomething: clips of human actions

Ego4D: egocentric daily life videos

BDD100K: driving dashcams

Waymo Open: driving camera videos

Video datasets sizes

Dataset	Domain	Ego	Pre	Bal	Annot	Avg. Dur (sec)	Dur (hr)	#Videos	Frame resolution
Diverse Pretraining									
Kinetics-400 (Kay et al., 2017)	Actions	✗	✓	✓	**Class**	10.2	851	400	340 × 255
WebVid-2M (Bain et al., 2021)	Open	✗	✓	✗	**Weak**	18	13k	–	320 × 240
HowTo100M (Miech et al., 2019)	Instructions	✗	✓	✗	**Weak**	4	135k	–	–
Egocentric									
Epic-Kitchens (Damen et al., 2022)	Cooking	✓	✗	✗	**Loc.**	510	100	37	1920 × 1080
Ego-4D (Grauman et al., 2022)	Daily	✓	✗	✗	**Loc.**	1446	120	931	1920 × 1080
Meccano (Ragusa et al., 2023)	Industry	✓	✗	✗	**Loc.**	1247	849	20	1920 × 1080
Assembly-101 (Sener et al., 2022)	Assembly	✓	✗	✗	**Loc.**	426	167	362	1920 × 1080
ImageNet-aligned									
R2V2 (Gordon et al., 2020)	ImageNet	✗	✓	✓	**Class**	–	–	–	467 × 280
VideoNet (Parthasarathy et al., 2022)	ImageNet	✗	✓	✓	**Class**	10	3055	–	
Walking Tours (ours)	Urban	✓	✓	✗	**None**	5880	23	10	3840 × 2160

For reference, ImageNet is 1.3M images of size ~470x390

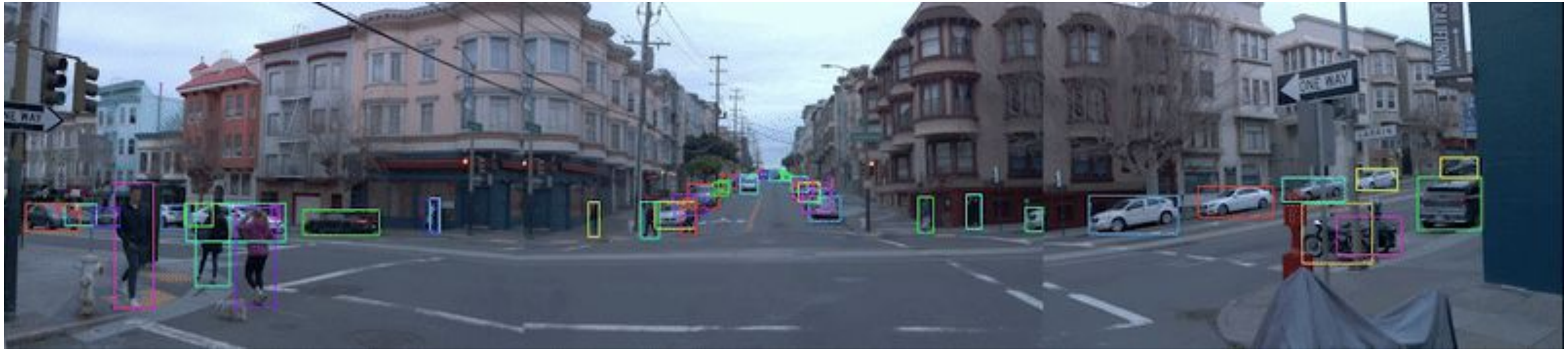
Example: WalkingTours

https://shashankvkt.github.io/dora

EPIC-Kitchens

https://epic-kitchens.github.io/

Example: Waymo Open



https://waymo.com/open/

Video Processing

Storage - time tradeoff:

1. Videos are compressed stacks of images
2. It takes time to decode videos into data arrays

Frame sampling

1. How much time in-between frames?
2. What level of temporal granularity do you care about?

Frame sampling

$\Delta t = 0$ (0s) $\Delta t = 15$ (0.5s) $\Delta t = 30$ (1s) $\Delta t = 45$ (1.5s)

Advanced: Adaptive Tokenization

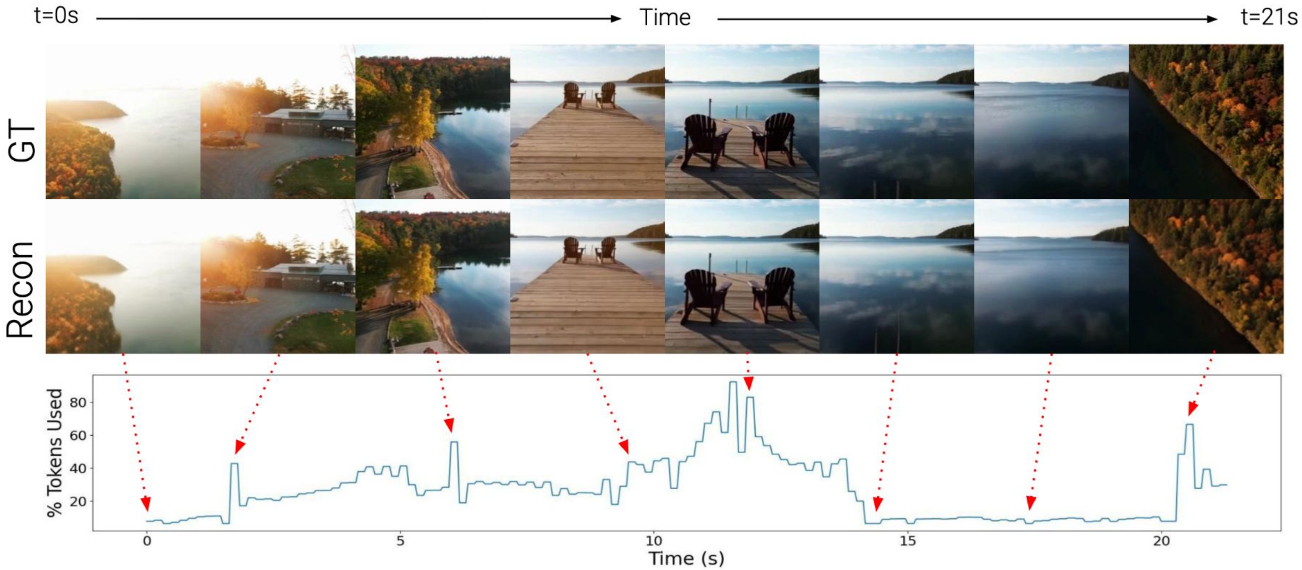


Figure 1 ElasticTok adaptively represent video based on information available. (Top) Ground-truth video frames. (Middle) Reconstructed frames with varying token usage. (Bottom) The bottom section depicts how ElasticTok dynamically adjusts token allocation over time, with the percentage of tokens used correlating to different content complexities in the video.

ElasticTok: Adaptive Tokenization for Image and Video

Advanced: Video Tokenization

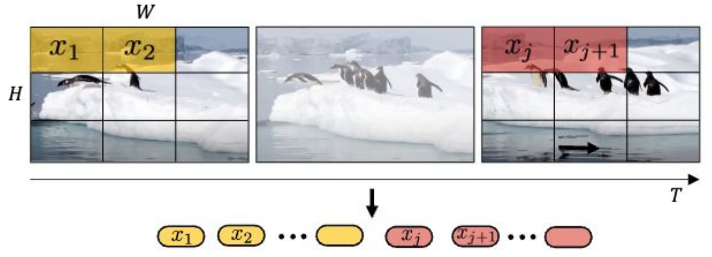


Figure 2: Uniform frame sampling: We simply sample n_t frames, and embed each 2D frame independently following ViT [18].

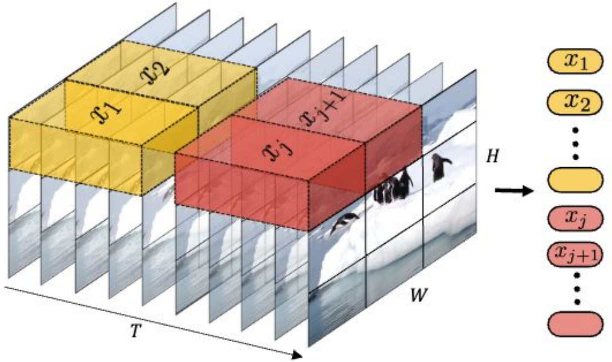
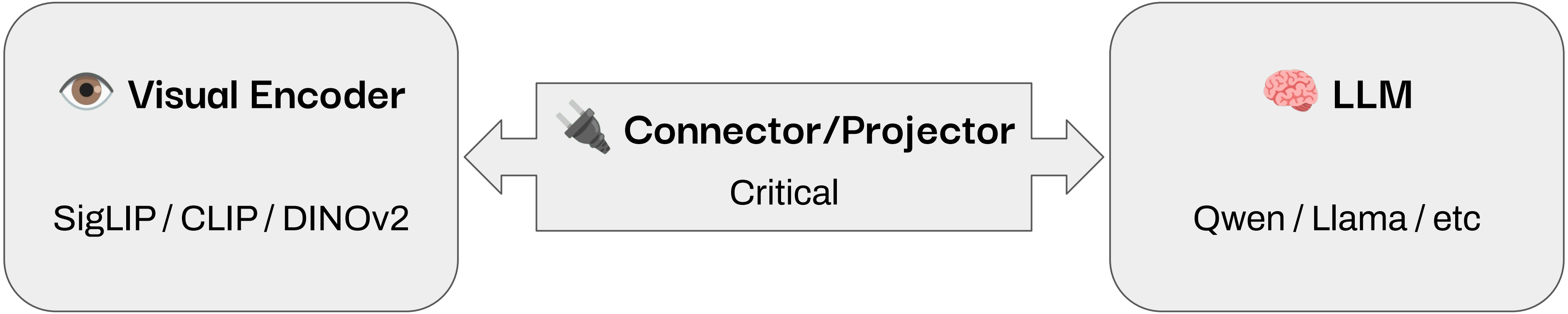


Figure 3: Tubelet embedding. We extract and linearly embed non-overlapping tubelets that span the spatio-temporal input volume.

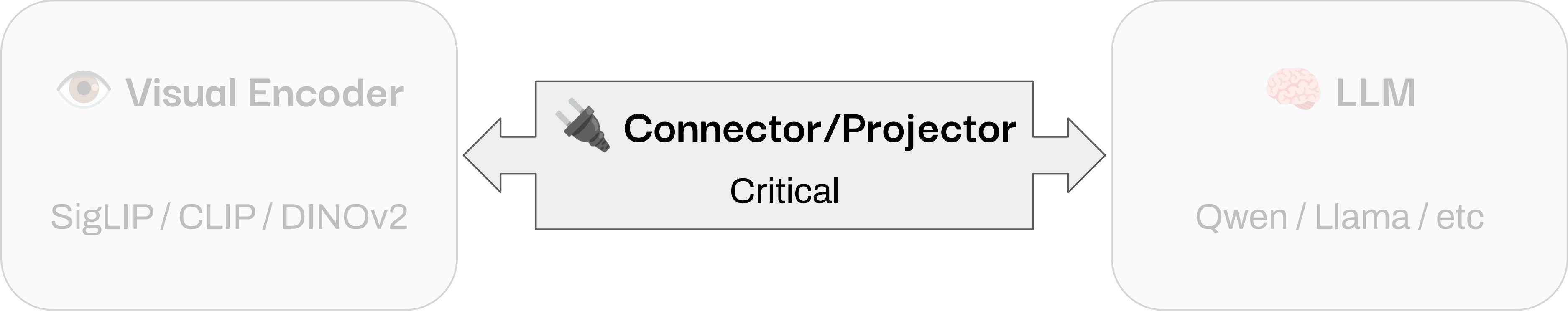
[[2103.15691] ViViT: A Video Vision Transformer]

III. Towards Video-LLMs

High-level of MLLMs



High-level of MLLMs



Q: How to effectively connect raw visuals w/ LLM?

Flamingo

- Very flexible!
- xattn to visuals
- Too heavy

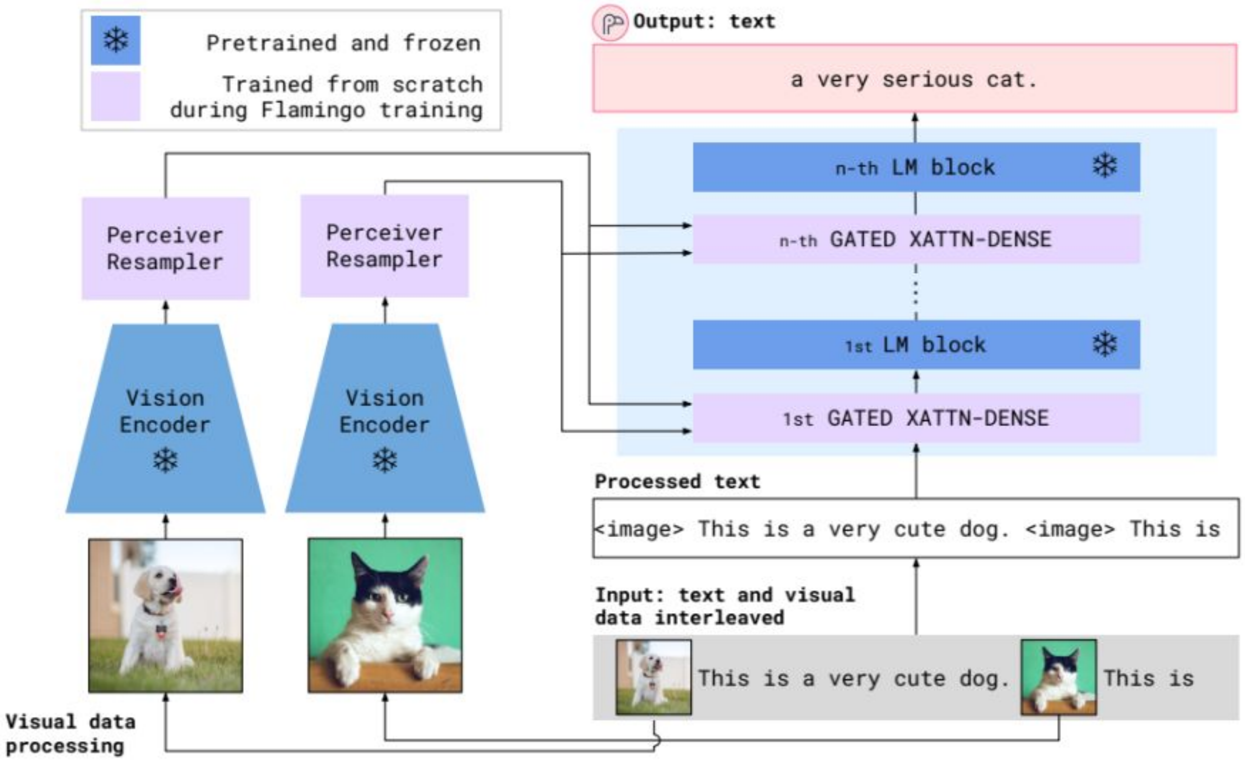


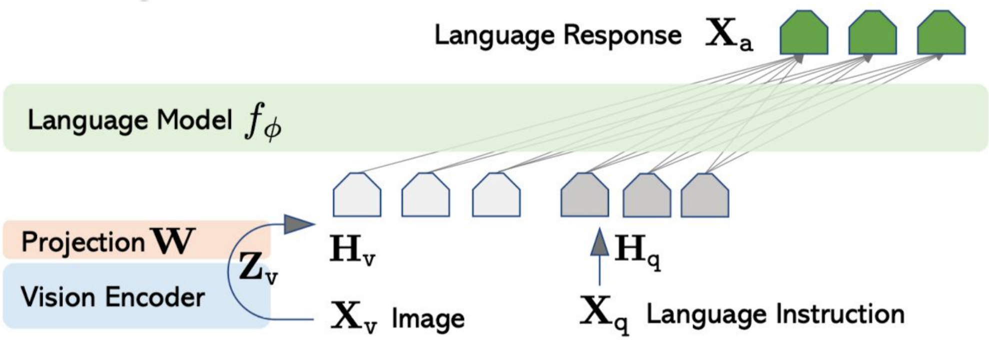
Figure 3 | **Overview of the Flamingo model.** The Flamingo models are a family of visual language model (VLM) that can take as input visual data interleaved with text and can produce free-form text as output. Key to its performance are novel architectural components and pretraining strategies described in Section 3.

[[2204.14198] Flamingo: a Visual Language Model for Few-Shot Learning]

LLaVA: Connect Single-Image to LLM

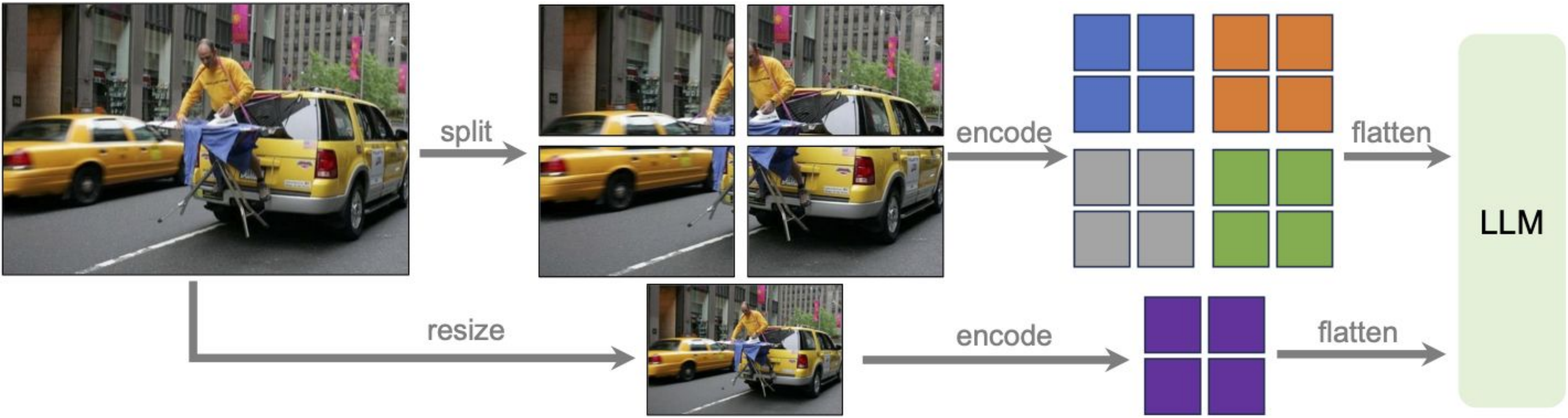
“Visual Instruction Tuning”

→ just tune the projector (linear layers). ***light!***



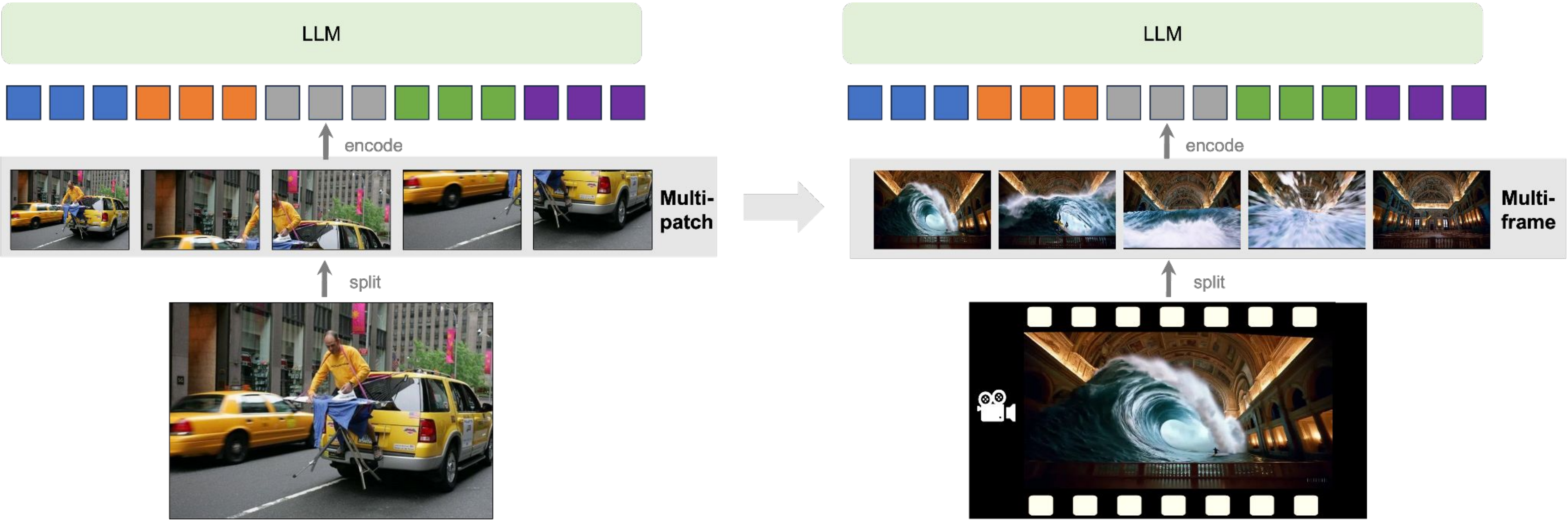
LLaVA | [2304.08485] Visual Instruction Tuning

High-res processing: pass multiple crops



[LLaVA-NeXT](#) | [[2310.03744] Improved Baselines with Visual Instruction Tuning](#)

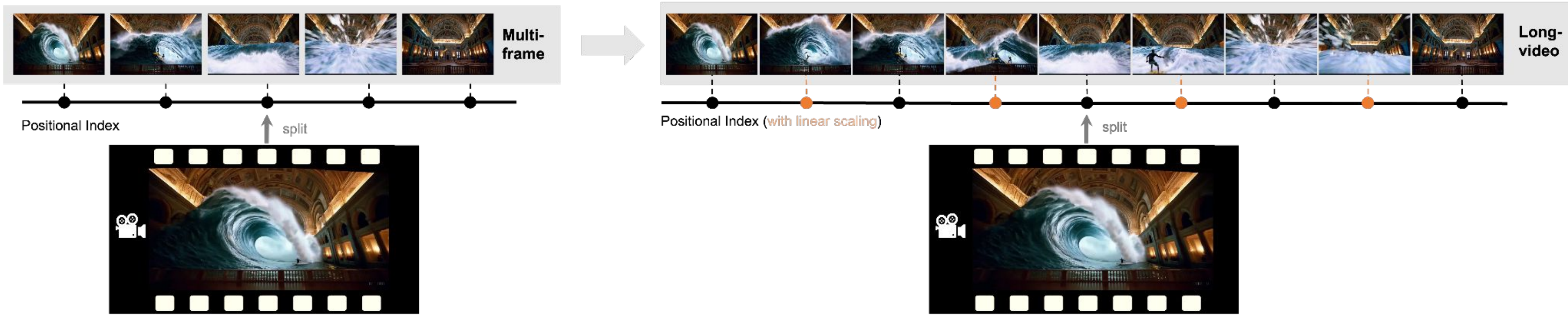
Multi-Crop (1 img) → Multi-*frame* (*video*)



[LLaVA-Video]

Multi-Crop (1 img) → Multi-***frame*** (***video***)

- Process more frames at test time
    - Miss fewer frames! But ***still*** drop many



Video-LLM Landscape

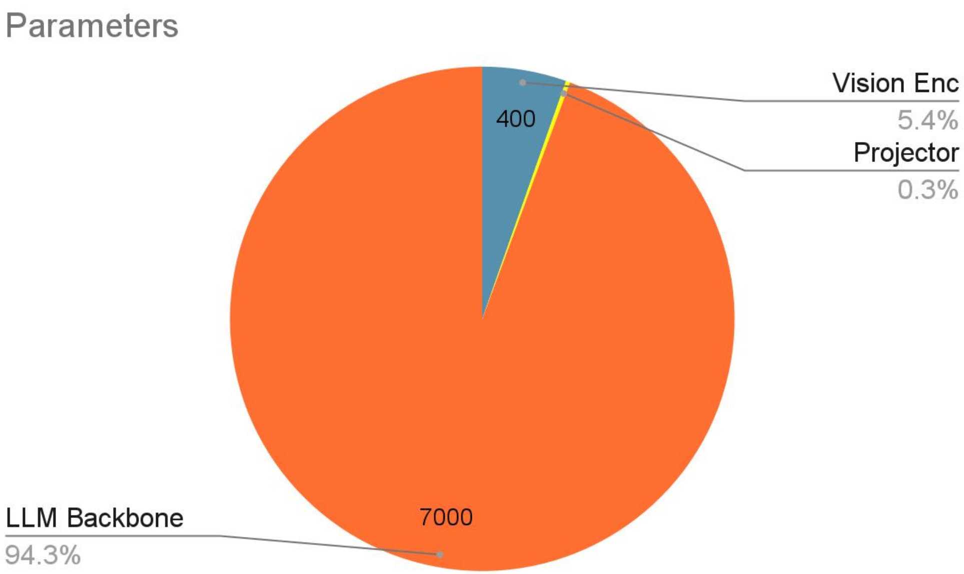
Organization	Models
Proprietary	
Google	Gemini 3.0 Pro, 2.5 Pro/Flash
OpenAI	GPT-5/5.2, GPT-4o
xAI	Grok 4/4.1
Anthropic	Claude 4.5 (frames only)
Reka	Core, Flash, Edge
Open — General	
Alibaba	Qwen3-VL, Qwen2.5-VL, QVQ-72B
OpenGVLab	InternVL3, InternVL 3.5
Meta	Llama 4 Scout/Maverick, Llama 3.2 Vision
Google	Gemma 3
Moonshot AI	Kimi-VL, Kimi-VL-Thinking
OpenBMB	MiniCPM-V 4.5, MiniCPM-o 2.6
Zhipu AI	GLM-4.6V
Microsoft	Phi-4 Multimodal
DeepSeek	DeepSeek-VL2, Janus-Pro
ByteDance	Seed1.5-VL, BAGEL
Open — Video-First	
LLaVA Labs	LLaVA-OneVision 1.5, LLaVA-Video
DAMO-NLP-SG	VideoLLaMA 3, VideoLLaMA 2
OpenGVLab	VideoChat-Flash, VideoChat2
ByteDance	Vidi 2.5
Meta et al.	Apollo
AI2	Molmo

IV. Practical Training

MLLM Training ≈ LLM Training

- LLM: 7B
- Vision: 400m
- Projector: ~20M

Good News: We can leverage the entire ecosystem of LLM optimization (FlashAttention, vLLM, FSDP, etc.)



Reducing Quadratic Cost of Attention: **Flash Attention**

Math:

- 1 Image ≈ 576 tokens.
- 1 min video (Sampled 1fps) = 60 images ≈ 34,560 tokens.
- Standard Attention cost: O(N^2)

Solution: **FlashAttention-2**

- What it is: *IO-aware* exact attention. Computes attentior by tiling blocks to avoid HBM (High Bandwidth Memory) reads/writes.
- Impact: Linear-scaling memory usage.
- Practical Advice: Try loading HF transformer model with **`attn_implementation="flash_attention_2"`** →→→

```python
import torch
from transformers import AutoModelForCausalLM

model = AutoModelForCausalLM.from_pretrained(
    "your-model-id",
    torch_dtype=torch.bfloat16
    attn_implementation="flash_attention_2"
).to("cuda")
```

Handling Large Batches:
Gradient Accumulation

Problem:

- Video-LLMs are VRAM-heavy. A single GPU often fits only `batch_size=1` or `2`.
- Small batches cause noisy gradients and unstable training.

Solution:

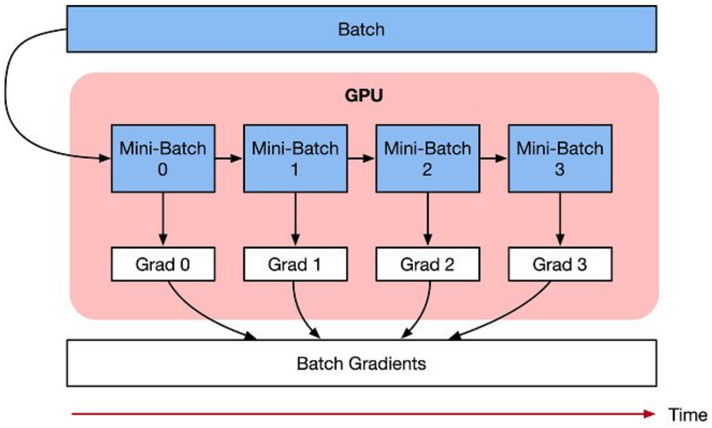
- **Virtual Batch Size:** Decouple your *hardware limit* from your *training batch size*.
- **Mechanism:**
  1. Forward/Backward pass on Micro-Batch 1 → **Do not clear grads**
  2. Repeat for ***N*** steps (accumulating gradients)
  3. optimizer.step() once.

Trade-off:

- ✅ **Memory:** Fits large logical batches on a single consumer GPU
- ❌ **Speed:** Sequential processing means it takes ***N*** times longer

Practical TIP: Let HF Accelerate handle accumulation for you!
→ Performing gradient accumulation with Accelerate

```
  from accelerate import Accelerator
- accelerator = Accelerator()
+ accelerator = Accelerator(gradient_accumulation_steps=2)
```




```
# Simulating batch_size = 64 with micro_batch = 8
accumulation_steps = 8

for i, batch in enumerate(dataloader):
    outputs = model(batch)
    loss = outputs.loss / accumulation_steps # Normalize!
    loss.backward() # Grads accumulate by default

    if (i + 1) % accumulation_steps == 0:
        optimizer.step() # Update once per 64 samples
        optimizer.zero_grad()
```

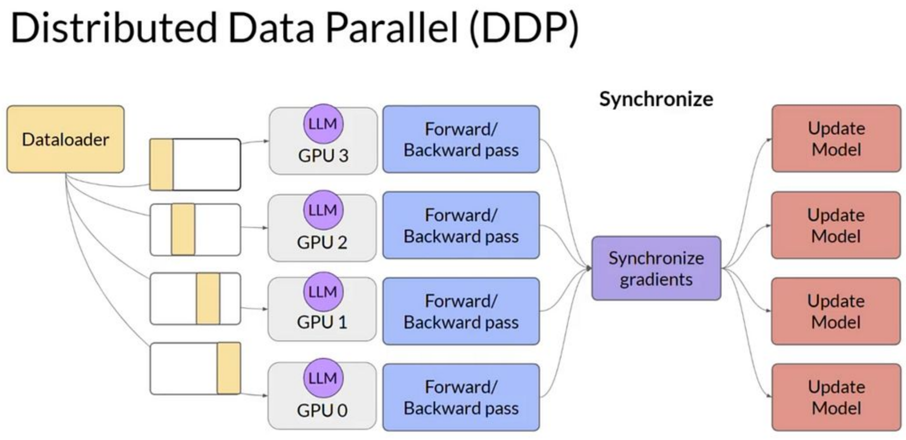
Scaling Across GPUs for Speed: **Distributed Data Parallel (DDP)**

Concept: "Parallel Gradient Accumulation"

Mechanism:
- **Replication:** Every GPU holds an identical copy of the *entire* model
- **Data Splitting:** The batch is split across GPUs (e.g., GPU 0 gets frames 1-4, GPU 1 gets frames 5-8)
- **Sync:** Gradients are averaged across GPUs before the update step

✅ **Pros:**
- **Speed:** Linear speedup (2 GPUs ≈ 2x faster)
- **Simplicity:** Minimal communication overhead

❌ **Cons:**
- **Memory Wall:** The *entire model* must fit on one GPU
- *Failure Mode:* You cannot train a 70B model on a 24GB card, no matter how many cards you have



Scaling Across GPUs for *Size*:
Fully-Sharded Data Parallel (FSDP) // DeepSpeed ZeRO

Concept: "Model Sharding"

The Problem: *What if the model itself is bigger than your VRAM?*

Solution (ZeRO Stage 3):

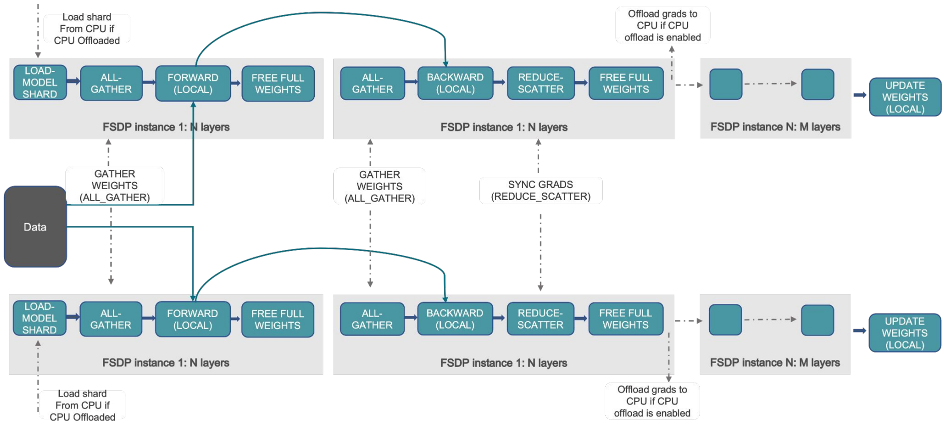
- **Shard Everything:** Split the Parameters, Gradients, and Optimizer States across all GPUs
- **On-Demand Fetching:** When GPU 0 needs Layer 1 to compute, it fetches those shards from other GPUs, computes, and discards them

✅ **Pros:**

- **Massive Scale:** Pools VRAM. 4x 24GB GPUs = ~96GB of usable Model Memory
- **Capability:** Enables fine-tuning 70B+ models on consumer clusters

❌ **Cons:**

- **Communication:** Heavy network traffic (constant swapping of shards)



Shrinking the Model: **Quantization**

Concept:

- Represent weights with lower precision to save VRAM
- **FP16 (Half Precision):** 2 bytes/param (Standard Inference)
- **INT4 (Quantized):** 0.5 bytes/param (Extreme Efficiency)

Example Impact (7B Model):

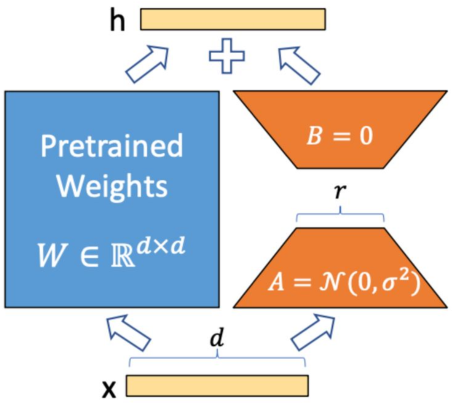
- **FP16:** ~14GB VRAM (Requires A100/A6000).
- **INT4:** ~4GB VRAM (Fits on consumer RTX cards).

The Stack:

- **`bitsandbytes`:** The standard library for 4-bit/8-bit loading in PyTorch
- **NF4 (Normal Float 4):** A data type optimized for neural network weights (preserves accuracy better than standard INT4)

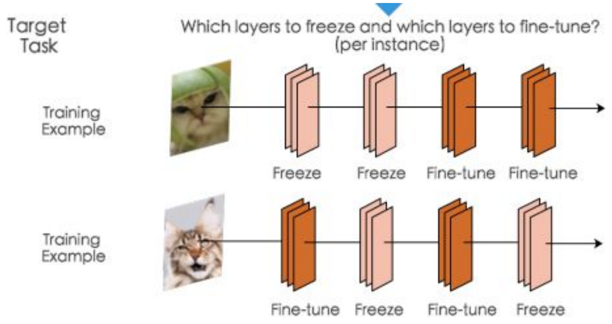
In Practice: can't train effectively in 4-bit (gradients vanish), so freeze the 4-bit weights and train a small adapter on top (QLoRA) →

Training on a Budget:
Parameter Efficient Finetuning (PEFT)



Partial Freezing

- Freeze Vision Encoder (Always).
- Freeze LLM / *some-most* layers
- Train only the Projector



LoRA (Low-Rank Adaptation)

- Don't update the full weight matrix W
- Train low-rank decomposition matrices A and B where ΔW = BA
- → Reduces trainable params by 10,000x

QLoRA (Quantized LoRA)

- Load the base LLM in 4-bit (NF4) **[frozen]**
- Attach LoRA adapters in 16-bit **[trainable]**
- → Finetune a 70B model on a single 48GB GPU (A6000)

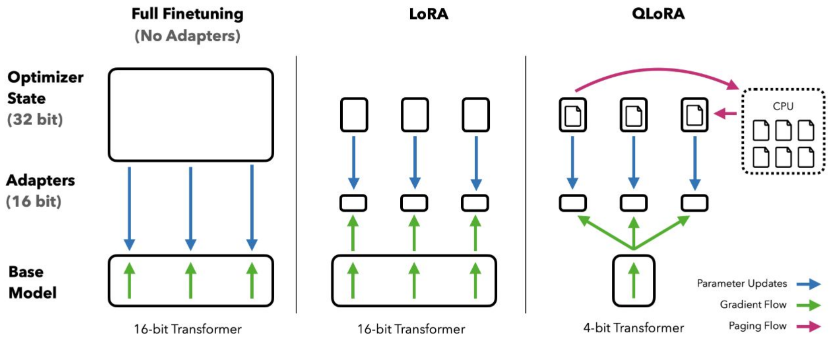


Figure 1: Different finetuning methods and their memory requirements. QLORA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

V. Failure Modes

Where Video-LLMs still break (non-exhaustive)

1. **Temporal confusion:** order, causality, “what changed?”
2. **Spatial grounding:** distances, collisions, occlusion, 3D ambiguity
3. **Long videos:** brute force attention scaling only goes so far
4. **Continual sensing:** tasks that require seeing *every* frame, fast responses

~all "Video" LLMs are secretly Image-LLMs in disguise!

→ frame subsampling is a persistent issue

To be continued…

Week 7 (Mar 3) — **Tutorial 5**