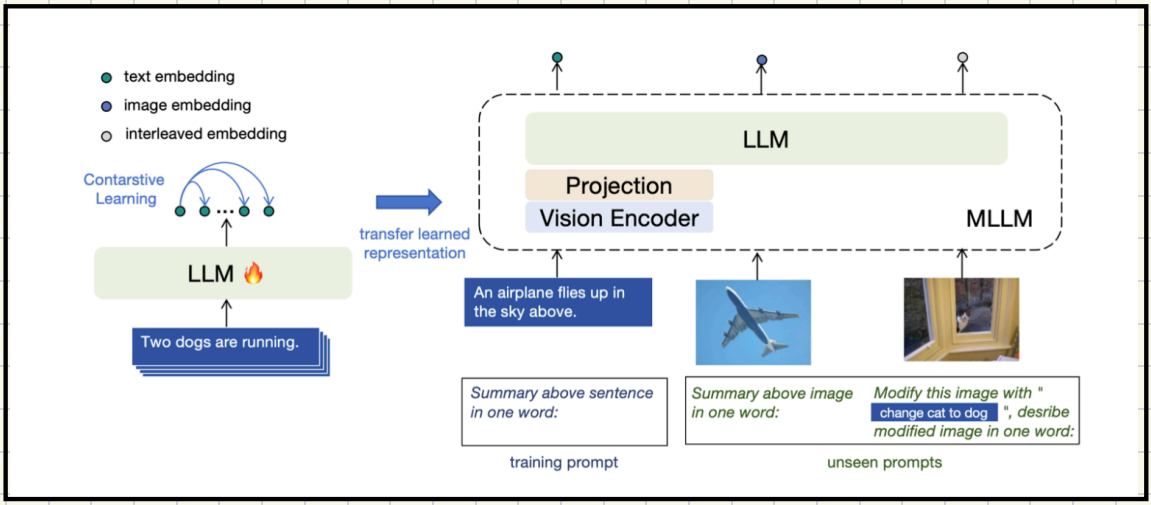
MindsOnML

https://www.linkedin.com/in/desaimann37/

https://github.com/desaimann37/MindsOnML

Hope this helps!

MLLM (Multimodal Large Language Models)



Disadvantages of previous work (CLIP models)

[Contrastive Language Image pre-training]

↳ (1) Seperate Encoders for image & text resulting in "Modality Gap".

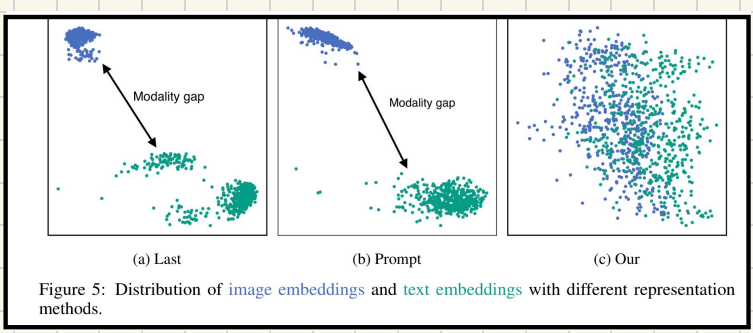


Figure 5: Distribution of image embeddings and text embeddings with different representation methods.

↳ (2) Cannot handle Interleaved Image/text Inputs.

Interleaved Input

Instead of this (non-interleaved):

vbnet

```
Text: "A dog is running in the park."
Image: dog.jpg
```

Interleaved input looks like:

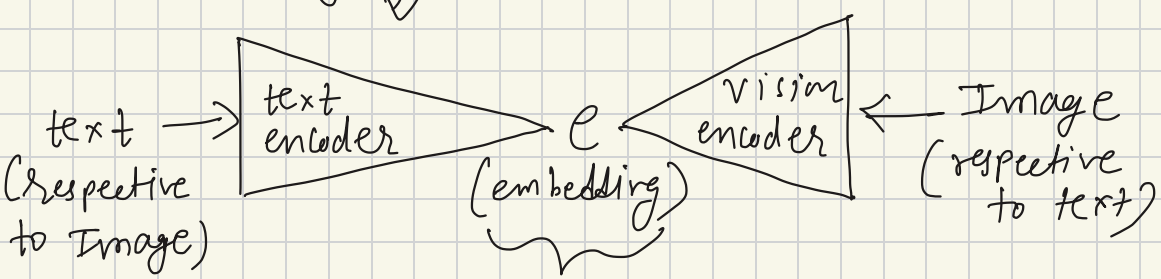
css

```
"The dog starts running" → [IMAGE_1] →
"then jumps over a log" → [IMAGE_2]
```

The model processes this as **one continuous sequence**.

(3) Needs massive Image-text pairs for training (~400 million pairs to train Standard CLIP models)

⇒ CLIP training



Same space embedding

- Here, both embeddings generates are of same dimension (R^d) but, they are different in values and to make in "same space", we need

Contrastive loss to calculate loss form image → text and text → image.

Step (1) Compute pairwise similarity.

$S_{ij} = v\_img\_i \cdot v\_txt\_j$

$(N \times N)$

for each img(i)

↳ it's matching text should have highest similarity.

Step (2) Loss for (image → text)

$L_{img} = \text{CrossEntropy}(\text{softmax}(S_{i,:}), \text{target} = i)$

- Loss for (text → image)

$L_{text} = \text{CrossEntropy}(\text{softmax}(S_{:,i}), \text{target} = i)$

Step (3) Final Loss:

$L = (L_{img} + L_{text})/2$

↳ this forces alignment.

- Till this point, CLIP processes img-text pairs and calculate loss (contrastive) like this and these disadvantages are along with them.
- So, we need to move to MLLMs to overcome these issues.

Simple Intuition of E5V (Using MLLMs):

(1) Force MLLM to convert any Input (Image OR text OR Image + text) into meaning based embeddings using prompts.

(2) Use prompts like:

Given Image: Summary of above image in one word.

Given Text: summary of text sentence in one word.

(3) Take last token embedding as the representation. This compresses meaning into a single token, not modality.

(4) Why this works: MLLM understands semantics not pixels or words.

- The prompt removes the "modality gap", hence text & image fall into same semantic space.
- Embeddings align by meaning, not the input type.

(5) Training strategy: (very important contribution)

- Train only on text pairs (images are optional to give)
- Use contrastive learning on text-text pairs (unlike, CLIP based models where image-text pairs are used)
- During training:
  - ↳ Remove visual encoder.
  - ↳ Train only the LLM.

(6) What E5V can do?

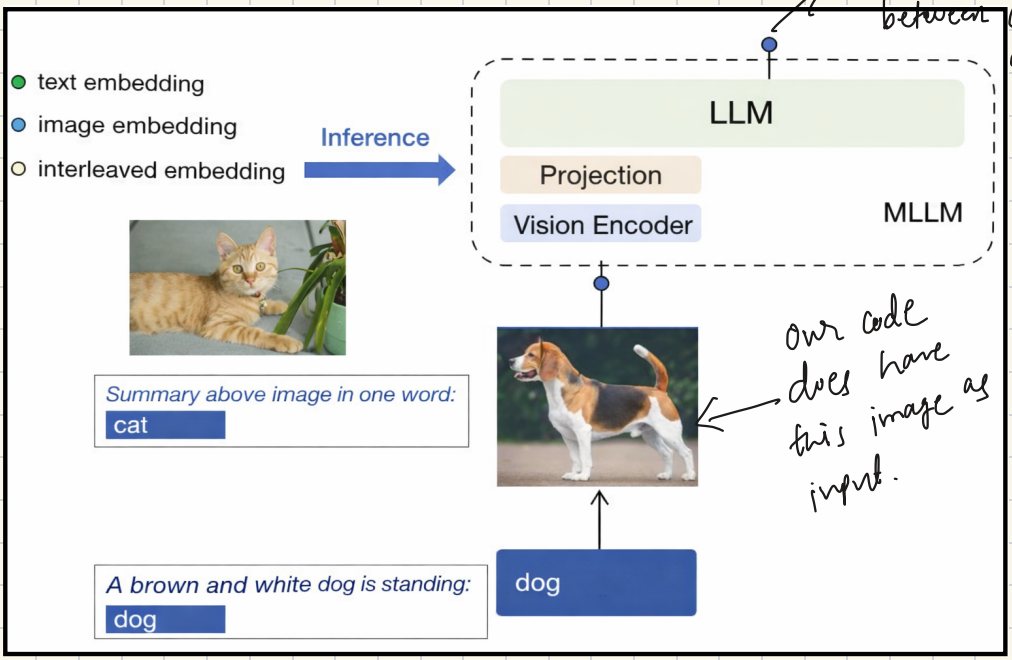
- Text-image Retrieval
- Image-Image Retrieval
- Composed Image (text + image) Retrieval
- Sentence Embeddings.
- All in zero-shot setting.

(7) At Inference time:

↳ Visual encoder & projector is available so, if we give image optionally, LLM will convert to text embeddings.

⇒ Results Obtained :

we use this to find cosine similarity between cat & dog.



our code does have this image as input.

GPU available

```
CUDA available: True
GPU count: 1
GPU name: NVIDIA A100-SXM4-80GB MIG 3g.40gb
Compute capability: (8, 0)
VRAM (GB): 42.144366592
(gclip) [mdesai23@gpu019 testing]$ python e5v.py
Loading processor...
Loading model...
Loading checkpoint shards: 100%| | 4/4 [00:04<00:00,  1.12s/it
Could not cache non-existence of file. Will ignore error and continue. Error: [Errno 122] Disk quota exceeded: '/home/mdesai23/.cache/huggingface/hub/models--ll
va-hf--llama3-llava-next-8b-hf/.no_exist/b041c0d0ea0dd0196d147206c210c8d1752fc2da/custom_generate'
Model loaded on: cuda:0
Vision encoder + projector frozen.

--- E5-V SANITY CHECK (INFERENCE) ---
Embedding shape: torch.Size([1, 4096])
sim(text dog, text animal): 0.89990234375
sim(text dog, text car): 0.79296875
sim(image dog, text dog): 0.63623046875
sim(image dog, image cat): 0.8779296875

E5-V image + text inference working correctly.
(gclip) [mdesai23@gpu019 testing]$
```

Final Result

LLAVA-NEXT (8B)

{ hugging face }

```
t1 = embed_text("A dog is running")
t2 = embed_text("An animal is moving fast")
t3 = embed_text("A parked car")
```

Training Data

- 558K filtered image-text pairs from LAION/CC/SBU, captioned by BLIP.
- 158K GPT-generated multimodal instruction-following data.
- 500K academic-task-oriented VQA data mixture.
- 50K GPT-4V data mixture.
- 40K ShareGPT data.

Additionally,

This python file (esv.py) shows only how given two images are similar (using cosine similarity)

- But, we can also use it for,

Contrastive question answering (QA)

```
image_labels = ["dog.jpg", "cat.jpg"]

# Contrastive questions
queries = [
 1. ("an image of a dog", "an image of a cat"),
 2. ("an image of a cat", "an image of a dog"),
 3. ("an image of an animal", "an image of a vehicle"),
    # Semantic logic
 4. ("an image of a pet", "an image of an object"),
 5. ("a living thing", "a non-living object"),
 6. ("a domestic animal", "a wild animal"),
]
```

Output : ① dog.jpg, ② cat.jpg, ③ dog.jpg, ④ cat.jpg, ⑤ cat.jpg, ⑥ dog.jpg

This gives image labels with matched result given by esv2.py

- All python files are attached in linked in post as well as on github Repo.

e5v.py:

```
import torch
import torch.nn.functional as F
from transformers import AutoProcessor, AutoModelForImageTextToText
from PIL import Image

# ============================================================
# CONFIG
# ============================================================

MODEL_ID = "llava-hf/llama3-llava-next-8b-hf"
DTYPE = torch.float16

# ============================================================
# PROMPTS (FROM E5-V PAPER)
# ============================================================

TEXT_PROMPT = """{text}
Summary of the above sentence in one word:"""

IMAGE_PROMPT = """<image>
Summary of the above image in one word:"""

# ============================================================
# LOAD MODEL
# ============================================================

print("Loading processor...")
processor = AutoProcessor.from_pretrained(
    MODEL_ID,
    use_fast=True
)

print("Loading model...")
model = AutoModelForImageTextToText.from_pretrained(
    MODEL_ID,
    dtype=DTYPE,
    device_map="auto"
)

model.eval()
print("Model loaded on:", next(model.parameters()).device)

# ============================================================
# FREEZE VISION ENCODER (PAPER REQUIREMENT)
# ============================================================

if hasattr(model, "vision_tower"):
    for p in model.vision_tower.parameters():
        p.requires_grad = False

if hasattr(model, "multi_modal_projector"):
    for p in model.multi_modal_projector.parameters():
        p.requires_grad = False

print("Vision encoder + projector frozen.")
```

```
# ============================================================
# EMBEDDING FUNCTIONS (INFERENCE ONLY)
# ============================================================

@torch.no_grad()
def embed_text(text: str) -> torch.Tensor:
    prompt = TEXT_PROMPT.format(text=text)

    inputs = processor(
        text=prompt,
        return_tensors="pt"
    ).to(model.device)

    outputs = model(
        **inputs,
        output_hidden_states=True,
        return_dict=True
    )

    emb = outputs.hidden_states[-1][:, -1, :]
    emb = F.normalize(emb, dim=-1)
    return emb


@torch.no_grad()
def embed_image(image: Image.Image) -> torch.Tensor:
    inputs = processor(
        images=image,
        text=IMAGE_PROMPT,
        return_tensors="pt"
    ).to(model.device)

    outputs = model(
        **inputs,
        output_hidden_states=True,
        return_dict=True
    )

    emb = outputs.hidden_states[-1][:, -1, :]
    emb = F.normalize(emb, dim=-1)
    return emb
```

```
# ============================================================
# SANITY CHECK (TEXT + IMAGE)
# ============================================================

if __name__ == "__main__":
    print("\n--- E5-V SANITY CHECK (INFERENCE) ---")

    # ---------- TEXT âï¸ TEXT ----------
    t1 = embed_text("A dog is running")
    t2 = embed_text("An animal is moving fast")
    t3 = embed_text("A parked car")

    print("Embedding shape:", t1.shape)
    print("sim(text dog, text animal):", F.cosine_similarity(t1, t2).item())
    print("sim(text dog, text car):", F.cosine_similarity(t1, t3).item())

    # ---------- IMAGE âï¸ TEXT ----------
    # Replace with your own image path
    img = Image.open("/home/mdesai23/project/MLLM/testing/cat_test.jpg").convert("RGB

    img_emb = embed_image(img)
    txt_emb = embed_text("a dog running")

    print("sim(image dog, text dog):",
          F.cosine_similarity(img_emb, txt_emb).item())

    # ---------- IMAGE âï¸ IMAGE (OPTIONAL) ----------
    img2 = Image.open("/home/mdesai23/project/MLLM/testing/dog_test.jpg").convert("RG

    img_emb2 = embed_image(img2)

    print("sim(image dog, image cat):",
          F.cosine_similarity(img_emb, img_emb2).item())

    print("\nE5-V image + text inference working correctly.")
```

e5v2.py :

```python
import torch
import torch.nn.functional as F
from transformers import AutoProcessor, AutoModelForImageTextToText

MODEL_ID = "llava-hf/llama3-llava-next-8b-hf"
DTYPE = torch.float16

# ============================================================
# PROMPTS (FROM E5-V PAPER)
# ============================================================

TEXT_PROMPT = """{text}
Summary of the above sentence in one word:"""

IMAGE_PROMPT = """<image>
Summary of the above image in one word:"""

# ============================================================
# LOAD MODEL
# ============================================================

print("Loading processor...")
processor = AutoProcessor.from_pretrained(
    MODEL_ID,
    use_fast=True
)

print("Loading model...")
model = AutoModelForImageTextToText.from_pretrained(
    MODEL_ID,
    dtype=DTYPE,
    device_map="auto"
)

model.eval()
print("Model loaded on:", next(model.parameters()).device)

# ============================================================
# FREEZE VISION ENCODER (PAPER REQUIREMENT)
# ============================================================

if hasattr(model, "vision_tower"):
    for p in model.vision_tower.parameters():
        p.requires_grad = False

if hasattr(model, "multi_modal_projector"):
    for p in model.multi_modal_projector.parameters():
        p.requires_grad = False

print("Vision encoder + projector frozen.")

# ============================================================
# EMBEDDING FUNCTIONS (INFERENCE ONLY)
# ============================================================

@torch.no_grad()
def embed_text(text: str) -> torch.Tensor:
    prompt = TEXT_PROMPT.format(text=text)

    inputs = processor(
        text=prompt,
        return_tensors="pt"
    ).to(model.device)

    outputs = model(
        **inputs,
        output_hidden_states=True,
        return_dict=True
    )

    emb = outputs.hidden_states[-1][:, -1, :]
    return F.normalize(emb, dim=-1)


@torch.no_grad()
def embed_image(image: Image.Image) -> torch.Tensor:
    inputs = processor(
        images=image,
        text=IMAGE_PROMPT,
        return_tensors="pt"
    ).to(model.device)

    outputs = model(
        **inputs,
        output_hidden_states=True,
        return_dict=True
    )

    emb = outputs.hidden_states[-1][:, -1, :]
    return F.normalize(emb, dim=-1)


# ============================================================
# CONTRASTIVE RETRIEVAL QA (FIXED OPTION A)
# ============================================================

def answer_question_contrastive(
    positive_query: str,
    negative_query: str,
    image_embeddings: torch.Tensor,
    image_labels: list
):
    """
    Contrastive retrieval:
    score = sim(pos_query, image) - sim(neg_query, image)
    """

    pos_emb = embed_text(positive_query)
    neg_emb = embed_text(negative_query)

    pos_scores = F.cosine_similarity(pos_emb, image_embeddings)
    neg_scores = F.cosine_similarity(neg_emb, image_embeddings)

    scores = pos_scores - neg_scores
    best_idx = scores.argmax().item()

    return image_labels[best_idx], scores[best_idx].item()

# ============================================================
# DEMO / SANITY CHECK
# ============================================================

if __name__ == "__main__":
    print("\n--- E5-V CONTRASTIVE RETRIEVAL QA DEMO ---")

    # Load images
    img_dog = Image.open(
        "/home/mdesai23/project/MLLM/testing/dog_test.jpg"
    ).convert("RGB")

    img_cat = Image.open(
        "/home/mdesai23/project/MLLM/testing/cat_test.jpg"
    ).convert("RGB")

    # Embed images
    emb_dog = embed_image(img_dog)
    emb_cat = embed_image(img_cat)

    image_embeddings = torch.cat([emb_dog, emb_cat], dim=0)
    image_labels = ["dog.jpg", "cat.jpg"]

    # Contrastive questions
    queries = [
        ("an image of a dog", "an image of a cat"),
        ("an image of a cat", "an image of a dog"),
        ("an image of an animal", "an image of a vehicle"),
        # Semantic logic
        ("an image of a pet", "an image of an object"),
        ("a living thing", "a non-living object"),
        ("a domestic animal", "a wild animal"),
    ]

    for pos_q, neg_q in queries:
        label, score = answer_question_contrastive(
            pos_q,
            neg_q,
            image_embeddings,
            image_labels
        )

        print(f"Positive query: {pos_q}")
        print(f"Negative query: {neg_q}")
        print(f"â Answer: {label} (score={score:.3f})\n")
```