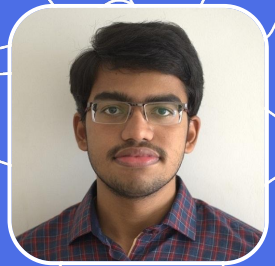

MULTIMODAL AI
SESSION #17
07th March 2025

About Us

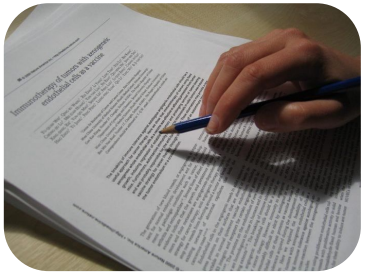
Henry Vo
@_lowkeyboi



Surya Guthikonda
@suryaguthikonda

What to expect?

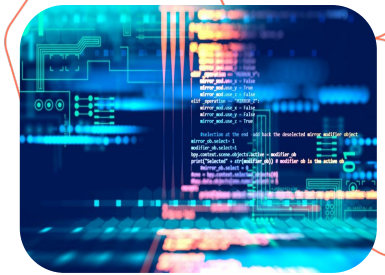
#multimodal-ml - Channel
@multimodal - Role



Paper Reading Session

Guest Speaker Session

Coding Session



Deep Dive Session

Second and Fourth
Friday 10 AM ET

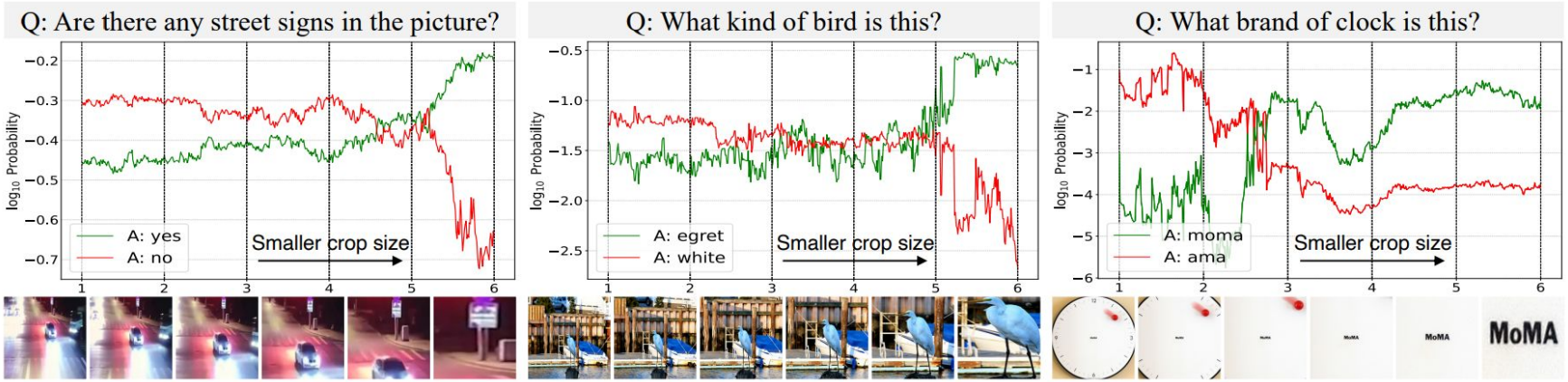
Paper Discussion Session

MLLMS Know Where To Look: Training-Free Perception of Small Visual Details with Multimodal LLMs

Cohere For AI

Motivation

BLIP2



Difficulty in Perceiving Small Visual Details

Perception Limitation or Localization Limitation

MLLMs' Sensitivity To the Size Of Visual Concepts

Quantitative Study using TextVQA Dataset (Validation)

Questions about small visual concepts
Image Ground truck BBox and Correct Textual Answer

Relative Size of the ground-truth Bounding box

$$S = \frac{A_{bb}}{A_{total}}$$

A_{bb} denotes the area of the ground-truth bounding box

A_{total} the total area of the image

1) S < 0.005 (small) [773]
2) 0.005 ≤ S < 0.05 (medium) [2411]
3) S ≥ 0.05 (large) [1186]

MLLMs' Sensitivity To the Size Of Visual Concepts

Model	Method	Answer Bbox Size (S)		
		small	medium	large
BLIP-2 (FlanT5$_{XL}$)	no cropping	12.13	19.57	36.32
	human-CROP	55.76	52.02	45.73
InstructBLIP (Vicuna-7B)	no cropping	21.79	30.58	45.30
	human-CROP	69.60	61.56	53.39
LLaVA-1.5 (Vicuna-7B)	no cropping	39.38	47.74	50.65
	human-CROP	69.95	65.36	56.96
Qwen-VL (Qwen-7B)	no cropping	56.42	65.09	68.60
	human-CROP	70.35	75.49	71.05
GPT-4o	no cropping	65.76	72.81	69.17
	human-CROP	75.63	81.32	71.72

Bias against Perceiving Smaller Visual Concepts

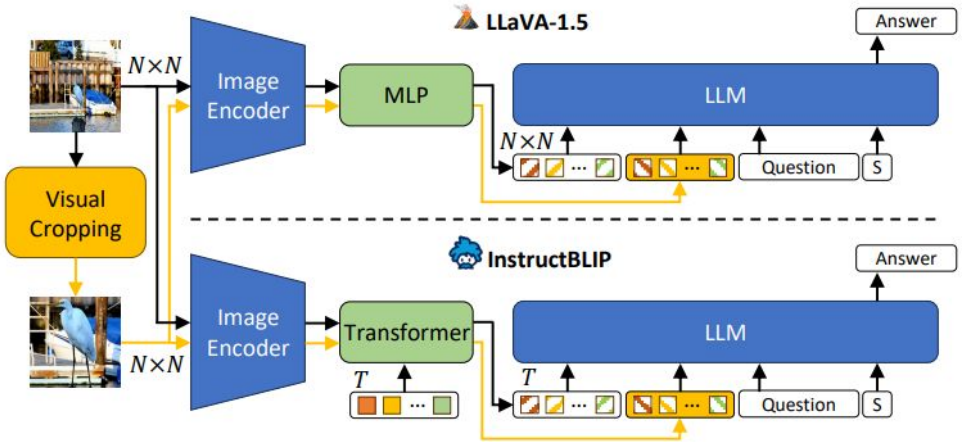
Perception Limitation Caused by Size of Visual Concepts

Visual Cropping Can Mitigate Perception Limitation

Do MLLMs Know Where to Look?

Experimental Setup

(x, q) image-question pair



LLaVA-1.5

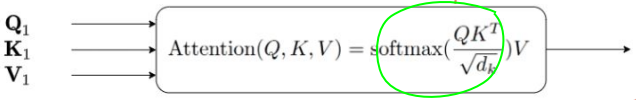
```
<image> USER:{question} Answer the question using a single
word or phrase.  ASSISTANT:
```

InstructBLIP

```
<image> Question:{question} Short Answer:
```

Do MLLMs' Know Where to Look?

Quantifying MLLMs' Spatial Attention over the Image



answer-to-token Attention

$A_{st}(x,q) \in \mathbb{R}^{L \times H \times 1 \times T}$

No. of Image Tokens

No. of Layers

Heads per Layer

$\hat{A}_{st}(x,q) = \frac{1}{H} \sum_{h=1}^{H} A_{st}(x,q)$

token-to-image Attention

$A_{ti} \in \mathbb{R}^{L_c \times H_c \times T \times N^2}$

No. of Image Patches

Connector

No. of Learnable Query Tokens

$\hat{A}_{ti}(x) = \frac{1}{H_c} \sum_{h=1}^{H_c} A_{ti}(x)$

Identity Tensor for LLaVA 1.5

answer-to-image Attention

$A_{si}(x,q) \in \mathbb{R}^{L \times L_c \times 1 \times N^2}$ where $A_{si}^{mk}(x,q) = \hat{A}_{st}^{m}(x,q)\hat{A}_{ti}^{k}(x)$

m, k - llm and connector layer indices

Relative Attention

$A_{rel}(x,q) = \frac{A_{si}(x,q)}{A_{si}(x,q')}$

q' ="Write a general description of the image."

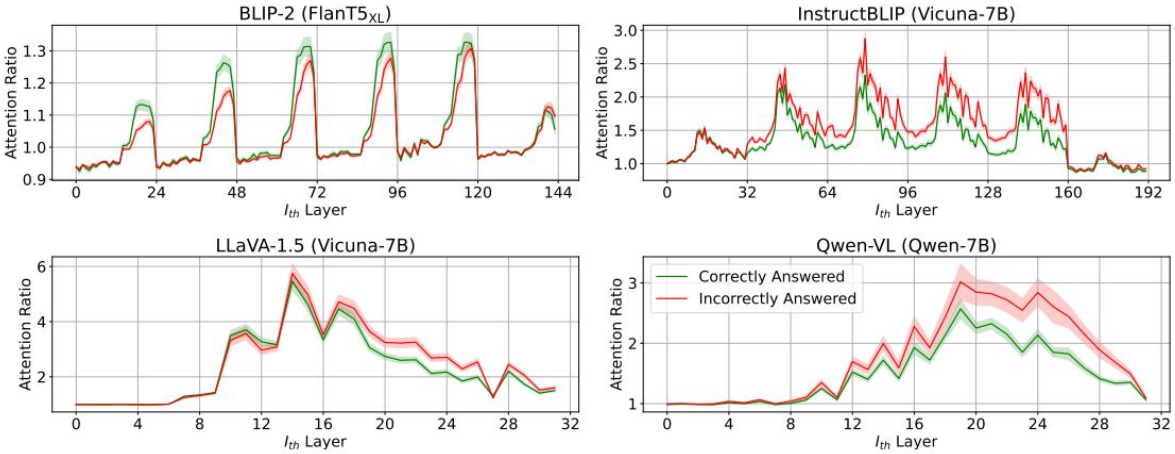
Transformers may use several tokens as registers to aggregate global information.

Do MLLMs' Know Where to Look?

Experiment on TextVQA (Validation)

Attention Ratio = Total relative attention inside the answer ground-truth bounding box / avg across all bounding boxes of same size

Measures MLLMs' attending to ground-truth bounding box



Models similarly strong attention to the correct region regardless of the correctness of answer

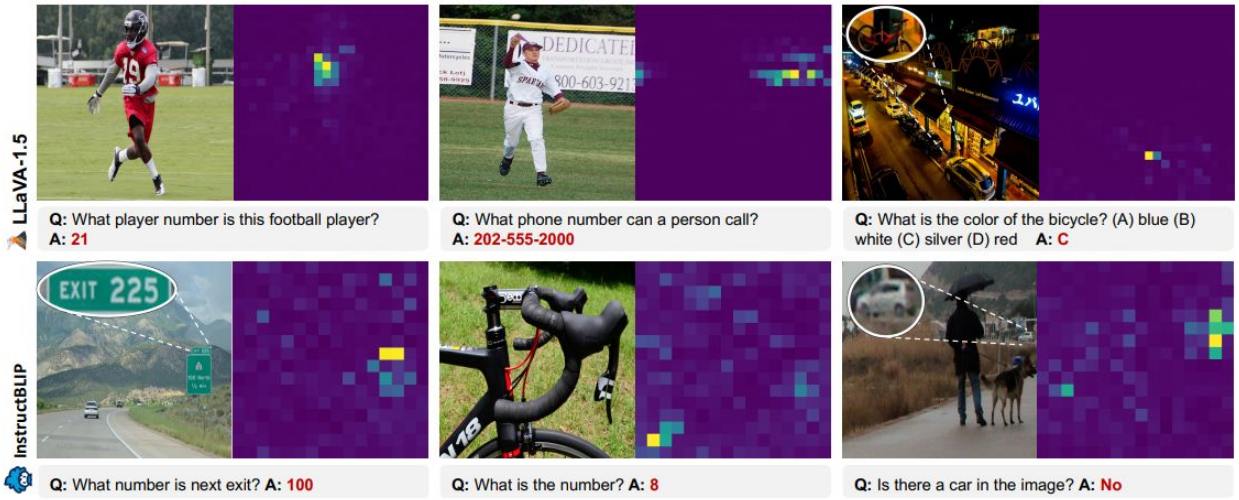
MLLMs tend to know where to look even if they answer incorrectly.

Localization Limitation ❌

Perception Limitation ✔

Do MLLMs' Know Where to Look?

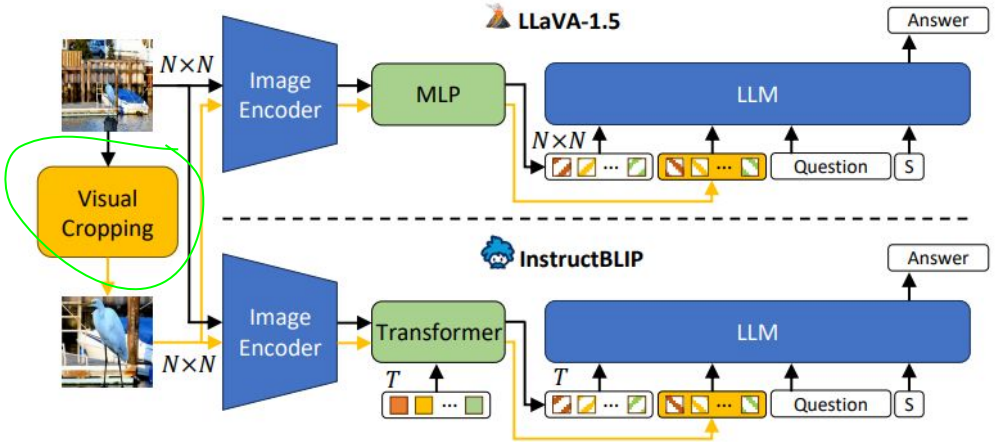
Yes, They Do 😀



Automatic Visual Cropping (VICrop)

To mitigate Perception Limitation one can train MLLMs with larger no. of image patches while maintaining per-patch resolution.

- *Increasing image resolution by a factor of α would require α^2N^2 input patches, dramatically increasing computational complexity (scaling as α^4N^4) for attention calculations.*



Training Free
and
Scalable to any image resolution

Relative Attention ViCrop (rel-att)

rel_attention_llava(image, prompt, general_prompt, model, processor):

```
# Prepare inputs for the prompt
inputs = processor(prompt, image, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)
pos = inputs['input_ids'][0].tolist().index(IMAGE_TOKEN_INDEX)

# Compute attention map for the 14th layer
att_map = model(**inputs, output_attentions=True)['attentions'][ATT_LAYER][0, :, -1, pos:pos+NUM_IMG_TOKENS].mean(dim=0).to(torch.float32).detach().cpu().numpy().reshape(NUM_PATCHES, NUM_PATCHES)

# Prepare inputs for the general prompt
general_inputs = processor(general_prompt, image, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)
general_pos = general_inputs['input_ids'][0].tolist().index(IMAGE_TOKEN_INDEX)

# Compute general attention map for the 14th layer
general_att_map = model(**general_inputs, output_attentions=True)['attentions'][ATT_LAYER][0, :, -1, general_pos:general_pos+NUM_IMG_TOKENS].mean(dim=0).to(torch.float32).detach().cpu().numpy().reshape(NUM_PATCHES, NUM_PATCHES)
# Normalize attention map
att_map = att_map / general_att_map
```

Relative Attention ViCrop (rel-att)

```
# Prepare inputs
inputs = processor(images=image, text=prompt, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)
general_inputs = processor(images=image, text=general_prompt, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)

outputs = model(**inputs, output_attentions=True)
general_outputs = model(**general_inputs, output_attentions=True)

# Extract attention maps
q_former_atts = outputs.qformer_outputs.cross_attentions
lm_atts = outputs.language_model_outputs.attentions

general_q_former_atts = general_outputs.qformer_outputs.cross_attentions
general_lm_atts = general_outputs.language_model_outputs.attentions

# Process Query Former attention maps (4th layer)
q_former_att = q_former_atts[QFORMER_LAYER][0, :, :, 1:].mean(dim=0).unsqueeze(0)
general_q_former_att = general_q_former_atts[QFORMER_LAYER][0, :, :, 1:].mean(dim=0).unsqueeze(0)

# Process Language Model attention maps (15th layer)
lm_atts = lm_atts[LM_LAYER][0, :, -1, :NUM_IMG_TOKENS].mean(dim=0).unsqueeze(0).unsqueeze(0)
general_lm_atts = general_lm_atts[LM_LAYER][0, :, -1, :NUM_IMG_TOKENS].mean(dim=0).unsqueeze(0).unsqueeze(0)

# Compute combined attention maps
att = torch.bmm(lm_atts, q_former_att).squeeze(1)
general_att = torch.bmm(general_lm_atts, general_q_former_att).squeeze(1)

# Compute attention map ratio
att_map = att / general_att

# Convert attention map to numpy and reshape
att_map = att_map.to(torch.float32).detach().cpu().numpy().reshape(NUM_PATCHES, NUM_PATCHES)
```

rel_attention_blip(image, prompt, general_prompt, model, processor):

Gradient Weighted Attention ViCrop (grad-att)

gradient_attention_llava(image, prompt, general_prompt, model, processor):

```
# Prepare inputs
inputs = processor(prompt, image, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)
pos = inputs['input_ids'][0].tolist().index(IMAGE_TOKEN_INDEX)

# Compute loss
outputs = model(**inputs, output_attentions=True)
CE = nn.CrossEntropyLoss()
zero_logit = outputs.logits[:, -1, :]
true_class = torch.argmax(zero_logit, dim=1)
loss = -CE(zero_logit, true_class)

# Compute attention and gradients
attention = outputs.attentions[ATT_LAYER]
grads = torch.autograd.grad(loss, attention, retain_graph=True)
grad_att = attention * F.relu(grads[0])

# Compute the attention maps
att_map = grad_att[0, :, -1, pos:pos+NUM_IMG_TOKENS].mean(dim=0).to(torch.float32).detach().cpu().numpy().reshape(NUM_PATCHES, NUM_PATCHES)
```

Gradient Weighted Attention ViCrop (grad-att)

```
# Prepare inputs
inputs = processor(images=image, text=prompt, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)
outputs = model(**inputs, output_attentions=True)

# Compute logits and loss
CE = nn.CrossEntropyLoss()
zero_logit = outputs.logits[:, -1, :]
true_class = torch.argmax(zero_logit, dim=1)
loss = -CE(zero_logit, true_class)


# Extract attention maps
q_former_atts = outputs.qformer_outputs.cross_attentions
lm_atts = outputs.language_model_outputs.attentions

# Compute gradients
q_former_grads = torch.autograd.grad(loss, q_former_atts, retain_graph=True)
lm_att_grads = torch.autograd.grad(loss, lm_atts, retain_graph=True)

# Process Query Former attention maps
q_former_att = q_former_atts[QFORMER_LAYER][0, :, :, 1:]
q_former_grad = q_former_grads[QFORMER_LAYER][0, :, :, 1:]
q_former_grad_att = (q_former_att * F.relu(q_former_grad)).mean(dim=0).unsqueeze(0)


# Process Language Model attention maps
lm_att = lm_atts[LM_LAYER][0, :, -1, :NUM_IMG_TOKENS]
lm_grad = lm_att_grads[LM_LAYER][0, :, -1, :NUM_IMG_TOKENS]
lm_grad_att = (lm_att * F.relu(lm_grad)).mean(dim=0).unsqueeze(0).unsqueeze(0)


# Compute combined attention map
att_map = torch.bmm(lm_grad_att, q_former_grad_att).squeeze(1).to(torch.float32).detach().cpu().numpy().reshape(NUM_PATCHES, NUM_PATCHES)
```

gradient_attention_blip(
image, prompt,
general_prompt, model,
processor):

Input Gradient ViCrop (pure-grad)

pure_gradient_llava(image, prompt, general_prompt, model, processor):

```
# Process inputs
inputs = processor(prompt, image, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)
general_inputs = processor(general_prompt, image, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)

# Apply high pass filter
high_pass = high_pass_filter(image, IMAGE_RESOLUTION, reduce=False)

# Enable gradients
inputs['pixel_values'].requires_grad = True
general_inputs['pixel_values'].requires_grad = True

# Initialize loss criterion
criterion = nn.CrossEntropyLoss()

# Forward pass for inputs
zero_logit = model(**inputs, output_hidden_states=False).logits[:, -1, :]
true_class = torch.argmax(zero_logit, dim=1)
loss = -criterion(zero_logit, true_class)

# Compute gradients
grads = torch.autograd.grad(loss, inputs['pixel_values'], retain_graph=True)[0]
```

high_pass_filter(image, resolusion, km=7, kh=3, reduce=True)

```
image = TF.resize(image, (resolusion, resolusion))
image = TF.to_tensor(image).unsqueeze(0)
l = TF.gaussian_blur(image, kernel_size=(kh, kh)).squeeze().detach().cpu().numpy()
h = image.squeeze().detach().cpu().numpy() - l
h_brightness = np.sqrt(np.square(h).sum(axis=0))
h_brightness = median_filter(h_brightness, size=km)
if reduce:
    h_brightness = block_reduce(h_brightness, block_size=(14, 14), func=np.sum)
```

Input Gradient ViCrop (pure-grad)

pure_gradient_llava(image, prompt, general_prompt, model, processor):

```
# Forward pass for general_inputs
general_zero_logit = model(**general_inputs, output_hidden_states=False).logits[:, -1, :]
general_true_class = torch.argmax(general_zero_logit, dim=1)
general_loss = -criterion(general_zero_logit, general_true_class)

# Compute general gradients
general_grads = torch.autograd.grad(general_loss, general_inputs['pixel_values'], retain_graph=True)[0]

# Process gradients
grads = grads.to(torch.float32).detach().cpu().numpy().squeeze().transpose(1, 2, 0)
general_grads = general_grads.to(torch.float32).detach().cpu().numpy().squeeze().transpose(1, 2, 0)

# Compute gradient norms
grad = np.linalg.norm(grads, axis=2)
general_grad = np.linalg.norm(general_grads, axis=2)

# Normalize and apply high pass filter
grad = grad / general_grad
high_pass = high_pass > np.median(high_pass)
grad = grad * high_pass

# Reduce gradient block size
grad = block_reduce(grad, block_size=(PATCH_SIZE, PATCH_SIZE), func=np.mean)
```

Input Gradient ViCrop (pure-grad)

pure_gradient_blip(image, prompt, general_prompt, model, processor):

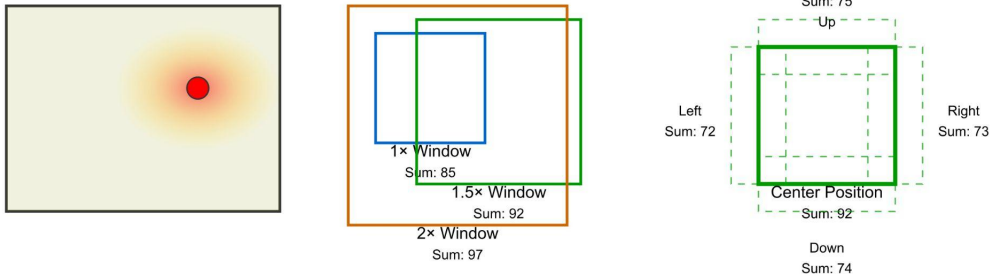
```
# Process inputs
inputs = processor(images=image, text=prompt, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)
general_inputs = processor(images=image, text=general_prompt, return_tensors="pt", padding=True).to(model.device, torch.bfloat16)

# Apply high pass filter
high_pass = high_pass_filter(image, IMAGE_RESOLUTION, reduce=False)

# Enable gradients
inputs['pixel_values'].requires_grad = True
general_inputs['pixel_values'].requires_grad = True

# Initialize loss criterion
criterion = nn.CrossEntropyLoss()

# Forward pass for inputs
outputs = model(**inputs, output_hidden_states=False)
zero_logit = outputs.logits[:, -1, :]
true_class = torch.argmax(zero_logit, dim=1)
loss = -criterion(zero_logit, true_class)

# Compute gradients
grads = torch.autograd.grad(loss, inputs['pixel_values'], retain_graph=True)[0]
```

Input Gradient ViCrop (pure-grad)

pure_gradient_blip(image, prompt, general_prompt, model, processor):

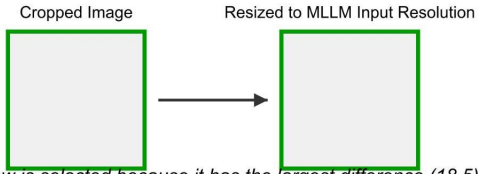
```
# Forward pass for general_inputs
general_outputs = model(**general_inputs, output_hidden_states=False)
general_zero_logit = general_outputs.logits[:, -1, :]
general_true_class = torch.argmax(general_zero_logit, dim=1)
general_loss = -criterion(general_zero_logit, general_true_class)

# Compute general gradients
general_grads = torch.autograd.grad(general_loss, general_inputs['pixel_values'], retain_graph=True)[0]

# Process gradients
grads = grads.to(torch.float32).detach().cpu().numpy().squeeze().transpose(1, 2, 0)
general_grads = general_grads.to(torch.float32).detach().cpu().numpy().squeeze().transpose(1, 2, 0)

# Compute gradient norms
grad = np.linalg.norm(grads, axis=2)
general_grad = np.linalg.norm(general_grads, axis=2)

# Normalize and apply high pass filter
grad = grad / general_grad
high_pass = high_pass > np.median(high_pass)
grad = grad * high_pass

# Reduce gradient block size
grad = block_reduce(grad, block_size=(PATCH_SIZE, PATCH_SIZE), func=np.mean)
```

Bounding Box Selection for ViCrop



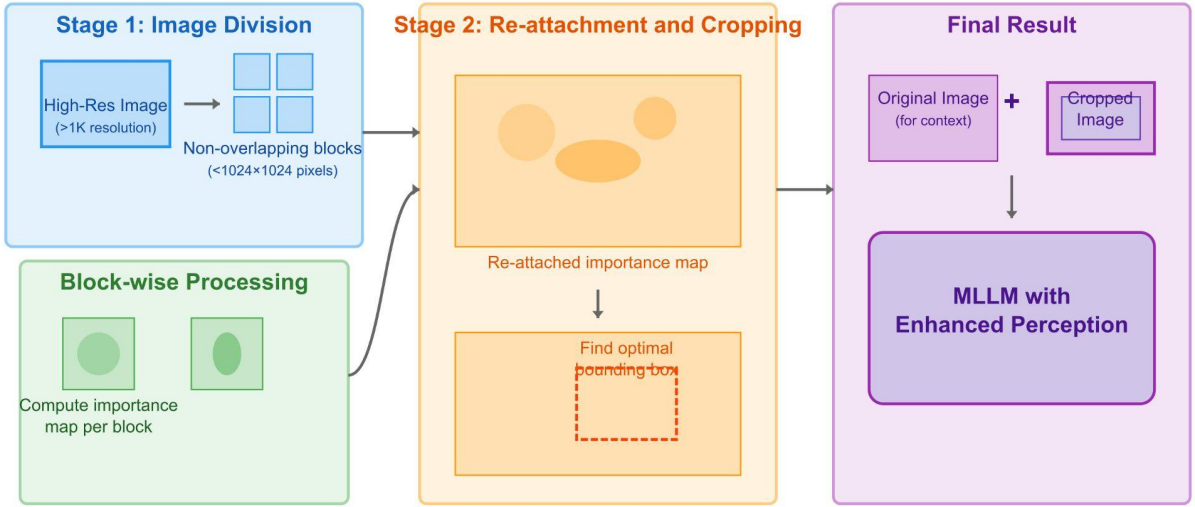
4. Final Selection and Output

Window Size	Best Sum	Avg. Adjacent Sum	Difference
1×	85	80	5
1.5×	92	73.5	**18.5**
2×	97	94	3

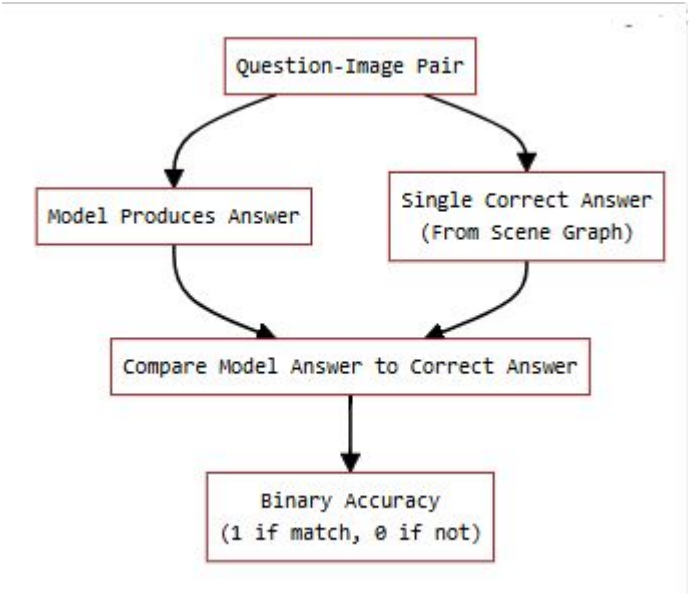


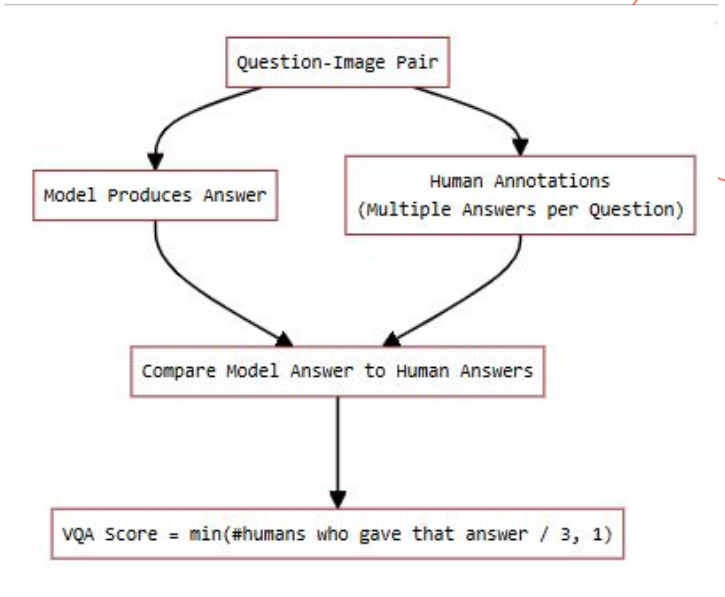
The 1.5× window is selected because it has the largest difference (18.5) between

High Resolution ViCrop



GQA Vs VQA Score





Accuracy of ViCrop on VQA Benchmarks

Model		Smaller Visual Concepts				Larger Visual Concepts		
		TextVQA†	V*	POPE	DocVQA	AOKVQA	GQA	VQAv2
LLaVA-1.5	no cropping	47.80	42.41	85.27	15.97	59.01	60.48	75.57
	rel-att	55.17	**62.30**	**87.25**	19.63	**60.66**	60.97	**76.51**
	grad-att	**56.06**	57.07	87.03	**19.84**	59.94	**60.98**	76.06
	pure-grad	51.67	46.07	86.06	17.70	59.92	60.54	75.94
InstructBLIP	no cropping	33.48	35.60	84.89	9.20	60.06	49.41	76.25
	rel-att	45.44	**42.41**	86.64	9.95	61.28	49.75	**76.84**
	grad-att	**45.71**	37.70	**86.99**	**10.81**	**61.77**	**50.33**	76.08
	pure-grad	42.23	37.17	86.84	8.99	61.60	50.08	76.71

- All ViCrop methods significantly improve MLLMs' performance on datasets with smaller visual details (TextVQA, V*) without requiring any training.
- The rel-att and grad-att methods consistently outperform pure-grad across most benchmarks.
- LLaVA-1.5 shows larger improvements from ViCrop methods than InstructBLIP, particularly on TextVQA and V*.
- ViCrop maintains or slightly improves performance on larger visual concept benchmarks (AOKVQA, GQA, VQAv2), showing no trade-off between small and large detail perception.

Choice of Layer and High-Res ViCrop

Model		Choice of Layer			High-Resolution ViCrop		
		Selective	Average	Δ	w/ High-Res	w/o High-Res	Δ
LLaVA-1.5	no cropping	47.80	–	–	42.41	42.41	–
	`rel-att`	55.17	55.45	+0.28	62.30	47.64	-14.66
	`grad-att`	56.06	56.26	+0.20	57.07	49.74	-7.33
	`pure-grad`	51.67	–	–	46.07	45.03	-1.04
InstructBLIP	no cropping	33.48	–	–	35.60	35.60	–
	`rel-att`	45.44	44.40	-1.04	42.41	38.74	-3.67
	`grad-att`	45.71	44.98	-0.73	37.70	42.41	+4.71
	`pure-grad`	42.23	–	–	37.17	42.41	+5.24

- For layer selection, averaging all layers works nearly as well as selecting specific layers, making it a good default choice when no training data is available.
- The high-resolution processing strategy significantly benefits LLaVA-1.5 (improving accuracy by up to ~15 percentage points) but can hurt performance for some methods on InstructBLIP.
- Even without optimal layer selection or high-resolution strategies, all ViCrop methods still improve baseline MLLM performance, demonstrating their robustness.
- Different models (LLaVA-1.5 vs InstructBLIP) respond differently to the same processing strategies, suggesting model-specific optimizations may be necessary.

External Tools and Inference Time

	Model	Original	SAM	YOLO	CLIP	rel-att	grad-att	pure-grad
Accuracy (TextVQA)	LLaVA-1.5	47.80	49.42	48.84	48.55	55.17	56.06	51.67
	InstructBLIP	33.48	39.23	36.49	39.61	45.44	45.71	42.23
CPU Time	LLaVA-1.5	2.26	91.53	0.97	5.46	14.43	11.33	29.86
	InstructBLIP	0.66				4.35	3.78	7.04
GPU Time	LLaVA-1.5	0.17	3.33	0.35	1.07	1.16	0.89	2.36
	InstructBLIP	0.06				0.28	0.29	0.60

- Internal ViCrop methods (rel-att, grad-att) significantly outperform external tools (SAM, YOLO, CLIP) in terms of accuracy improvement for both LLaVA-1.5 and InstructBLIP models.
- The grad-att method provides the highest accuracy boost for both models (56.06% for LLaVA-1.5 and 45.71% for InstructBLIP).
- While most ViCrop methods add reasonable inference time overhead (1-2 seconds on GPU), SAM is significantly slower than other approaches.
- Internal ViCrop methods have a fixed time overhead regardless of answer length, unlike MLLMs whose inference time scales linearly with the number of tokens generated.
- The rel-att method offers an excellent balance between accuracy improvement and computational efficiency, requiring only about the time needed to generate 5 tokens.

Conclusion

- MLLMs struggle to perceive small visual details even when they correctly attend to the relevant image regions, but this limitation can be mitigated without additional training using the model's own attention patterns to guide visual cropping.

- The training-free visual cropping methods (ViCrop) significantly improve accuracy on small-detail tasks by leveraging a model's internal knowledge of where to look, demonstrating that MLLMs already "know where to look" even when they provide incorrect answers.

Limitations and Future Work

- ViCrop struggles with relation and counting questions because it can only focus on one image region at a time.
- While inference time overhead is reasonable (a few seconds), it could be optimized through techniques like lower precision and weight quantization.
- Integration with Matryoshka Query Transformer (MQT) could potentially reduce computational costs by enabling varying visual context sizes during inference.
- The different ViCrop methods show complementary benefits, suggesting potential improvements through combining approaches based on prediction uncertainty.

Breakout Session (10 min)

This paper focuses on small visual details, but what other perceptual limitations might exist in current MLLMs that could benefit from similar analysis? What methodologies would you propose to investigate these potential limitations?


THANK YOU :)