IVTP: Instruction-guided Visual Token Pruning

Figures

Fig. 1 – Different Token Pruning Ideas *(Slide 3)*

A) Trainable Compression
Pruning **AFTER** the ViT (After Visual Encoder, before Projection.)

1. ViT encodes full image into a set of tokens. (Patches)
2. Pruning / Compression module compresses tokens and removes redundant tokens.
3. Remaining tokens are passed through Projection Layer which converts the image language into a word language. (Ready for the LLM to interpret.)
4. LLM takes the question / prompt with the Image tokens into account and reasons. (Provides an Output)

Pros:
- Pruning Layer are Trainable (NOT FROZEN)
    - Learns to compress
- Reduces tokens before sending the LLM.

Cons:
- Trainable layer is specific to a single ViT + LLM combo. (Not transferable.)
    - Changing either LLM or ViT the module becomes useless.
    - Different Models expect different features. (Misalignment.)
- Token important is only evaluated visually. (Not taking into account instruction / prompt; while possible in their example it is not instruction aware.)
- The ViT still processes all tokens (Heavy computation in ViT.)

B) Vision-Only Pruning
Pruning **IN** the ViT (Integrated in the Visual Encoder.)

1. ViT encodes image into a set of tokens and throws out unnecessary tokens such as background or low-important patches. (Pruning in ViT layer/by/layer)
2. Remaining tokens are passed through Projection Layer which converts the image language into a word language. (Ready for the LLM to interpret.)
3. LLM takes the question / prompt with the Image tokens into account and reasons. (Provides an Output)

Pros:
- Reduces computation in the ViT via Pruning

Cons:
- Pruning Layers are NOT Trainable (FROZEN in ViT)
- Token important is only evaluated visually. (Not taking into account instruction / prompt)

C) Instruction-guided Visual Token Pruning (IVTP)
Pruning **IN** the ViT **AND IN** the LLM (Two-Stage Pruning System)

1. ViT encodes image into a set of tokens.
    a. Pruning #1 removes visually redundant or low-importance tokens using Group-wise Token Pruning (GTP).
2. Remaining tokens are passed through Projection Layer which converts the image language into a word language. (Ready for the LLM to interpret.)
3. Instruction-guided Component processes the users instruction/prompt through a CLIP text encoder.
    a. CLIP identifies which words in the instruction/prompt are related to the visual content and produces a pseudo-CLS token summarizing relevant instruction meaning.
4. LLM uses the pseudo-CLS token and performs pruning.
    a. Pruning #2 removes visual tokens that are not relevant to the instruction. (Keep “dog”, removes “sky” if the question is about a dog.)
    b. Using filtered tokens and instruction the LLM reasons. (Provides an Output)

Pros:
- Removes far more tokens while preserving accuracy as we will see later.
- Visual tokens assessed and reasoned on are related to user’s instruction/prompt. (Instruction-aware)
- Works with frozen ViT + frozen LLM
    - Uses attention maps + CLIP semantics (no retraining)
- Token important ranking in Group-Wise Token Pruning (GTP) aggregates attention across multiple layers. (Reducing noise.)

Cons:
- Argued more complex due to having 2 Pruning Stages
- CLIP text encoder required.
- Slight overhead inside LLM
    - But, less computation then processing all tokens.

Fig. 2 – IVTP Architecture *(Slide 4)*

1. Start w/ an Image. We pass the image into the Visual Encoder (which consists of Visual Transformer Layers (ViT’s)

2. The Visual Encoder splits the image into a ton of tiny patches.
3. Group-wise Token Pruning (GTP) evaluates token importance using attention rollout.
    a. Combines info from multiple transformer layers. (Specifically 3 layers.)
        i. Attention across several transformer layers for the Visual CLS token + Patch Tokens
        ii. Importance score is calculated.
    b. Tokens with low importance are removed. (Pruning #1)
4. Remaining tokens are passed through a Projection Layer which converts the image language into a word language. (Ready for the LLM to interpret.)
5. CLIP Text Tokenizer / CLIP Text Encoder (CLIP is trained on image and text alignment.) Finds what words are relevant to an image.
    a. Creates text embeddings for the instruction/prompt text, and any OCR text; it aligns it with the CLIP's vision space. (Passes into CLIP Text Encoder
    b. Computes relevance between each word from the instruction/prompt and the visual CLS token.
        i. "beer", "self" = high relevance
        ii. "what", "kind" = low relevance
    c. Creates a pseudo CLS token (visually relevant token information.)
        i. We will use this inside the LLM to guide pruning.
6. LLM Text Tokenizer
    a. LLM cannot interpret or read or make any sense of CLIP tokens.
    b. It takes the instruction/prompt and any OCR words and uses the CLIP tokens and tokenizes them into LLM readable tokens. (For LLM Part 1)
7. LLM (Part 1)
    a. Group-wise Token Pruning (GTP) runs again (Pruning #2), but is guided by the pseudo CLS NOT the visual CLS. Visual tokens are scored based on attention to the pseudo CLS token. Irrelevant visual tokens are pruned.
        i. "What kind of beer is this?" = Keep tokens around beer label
        ii. Remove background wall, table, sky, etc…
8. LLM (Part 2)
    a. After the 2 stages of pruning.
        i. Important visual tokens, instruction/prompt text, and pseudo CLS token go through reason and generate a final answer.

Group-wise Token Pruning

Takes in:

- Image Tokens
- CLS Token (Visual CLS token -or- pseudo CLS token)
    - If we are doing the GTP in Visual Encoder (uses Visual CLS token)
    - If we are doing the GTP in LLM Part 1 (uses pseudo CLS token)

- Textual tokens if we are doing the LLM Group-wise Token Pruning

Transformer layers:

- Computes self-attention (between every pair of tokens & CLS
- Creates a Attention Map (uses CLS)
    - Rows = Tokens Giving Attention
    - Columns = Tokens Receiving Attention
- Standard transformer operations (Add & norm) + Feed Forward Network

Every Group of 3 Transformer layers:

- Collects the 3 attention maps. (One from each layer.)
- Attention Rollout (Will go in depth later.)
    - Creates 1 Attention Map
- Patch Values (Show strong or meaningful each patches visual features are.)
    - Patch with bottle label, or text is high.
    - Patch with background is low.
- Multiplies them together to get a final importance score.
- Importance scores are ranked / sorted.
        - High = More Importance
        - Select top K tokens (Prune the others)
        - Low scoring tokens are removed. (Pruned)

Fig. 3 – Different Token Pruning Methods *(Slide 5)*

X-Axis = TFLOPS

- How much computation the model uses.

Y-Axis = Average Accuracy %

- How accurate the model is on all 12 evaluation benchmarks.

Goal:

- Show which methods gives the best accuracy while using the least compute.

Blue = IVTP (Ours)

- Has the highest accuracy at every compute level.
- Same compute budget (TFLOPS) as the others but IVTP is more accurate.
- Or for the same accuracy as the others, IVTP would require less compute.

Less compute (TFLOPS)	**More compute (TFLOPS)**
- More Pruning Occurs - Accuracy Drops	- Less Pruning Occcurs - Accuracy approaches full LLaVA baseline (no pruning)

LLaVA-1.5-7B uses 15.4 TFLOPs (compute) for ~62.7% accuracy
IVTP gets almost the same accuracy with less compute as low as ~11 TFLOPs (compute)

Summary:

- Competing methods of pruning fall behind
- Random = bad because it throws away tokens without any logic.
- When re sources are limited (low compute) others prune too aggressively or poorly. (Lower accuracies)

Fig. 4 – Performance based on Tokens Pruned *(Slide 6)*

X-Axis = TFLOPS
- How much computation the model uses.

Y-Axis = Average Accuracy %
- How accurate the model is on all 12 evaluation benchmarks.

Goal:
- Show how (3) different pruning methods perform as you vary the number of remaining visual tokens.
- 16, 32, 64, 128, 256, 512 tokens
- Identify which method maintains best accuracy for the least compute.

Blue = IVTP (Ours)
- Highest accuracy at every token level. (16 – 512)
- At the same compute, IVTP is always more accurate than ToMe & TopK.
- For the same accuracy, IVTP requires less compute.

Summary:
- IVTP consistently outperforms ToMe and TopK at every pruning level.
- IVTP may use slightly higher TFLOPs (compute) for the equivalent methods with the same # of remaining tokens but it has significantly higher accuracy.
- Others:
  - Lower TFLOPs (compute)
  - Lower accuracy
  - Unstable when pruning heavy (low TFLOPs, low compute)
  - Not instruction-aware at all.

Fig. 5 – Visualization *(Slide 7)*

Same Number of tokens in results.

- Original Image
- Result of TopK
  - Ignores instruction.
  - Keeps tokens with highest visual similarity to CLS.
  - Keeps irrelevant stuff and prunes important stuff.
- Result of IVTP
  - Instruction guided.
  - Keeps only tokens relevant to the instruction/prompt.
  - Removes things intelligently w/ logic.

Equations

Eq. 1 – Response Generation *(Slide 9)*

This equation describes how the LLM generates a final answer **after pruning**.

The model generates one token at a time, and each token depends on the image tokens, all previous words, and the model's parameters.

(autoregressive generation process)

$p(\mathbf{X})$	**Probability of the entire generated answer.** (How likely the model thinks the final sequence of words is.)
$\prod_{i=1}^{M}$	**Multiply the probability of each generated token** (Generate text one token at a time; get the full sentence probability via multiplication.) M = Total # of tokens in the generated answer
x_i^L	**The i'th language token the model generates.** (Example: "There are 3 clocks"; x1 = "There", …)
$x_1^V, \ldots, x_F^V$	**The visual tokens from the image.** The model generates e ach word based on the image. F = Total # of visual tokens (after pruning)
$x_1^L, \ldots, x_{i-1}^L$	**The model uses all earlier generated words to decide next word.** ("There are 3", next generated word is "Clocks".)
Θ	**All learned parameters of the L LM** (Weights of the model.)

Eq. 2 – Attention Weights *(Slide 10)*

This equation shows how a transformer calculates attention, specifically the attention map that tells the models which tokens should focus on which other tokens.

Used in **Visual Token Pruning (GTP), & LLM Instruction-guided Token Pruning (GTP)**.

A	**Attention Map** The matrix showing how each token looks at each other token. (Including itself)
QK^T	**Queries and Keys** Every token gets a Query vector and a key vector This computes similarity between every pair of tokens. (# tokens x # tokens) Large value = token cares about that token. *T = Transpose required to multiply Q and K.*
$\sqrt{d}$	**D = Dimension of the hidden vector** *(how many features each token has)* Prevents the dot product from getting to large. (Normalizes)

	Keeps attention scores stable.
A_{mask}	**Adds a Mask** - In the LLM: Block attention to future tokens (causal mask) ○ Prevents from looking at future words. Predicting word #5 can't look at language token 6,7,8. - In ViT: No mask all tokens can be seen. You cannot attend to illegal positions.
$\text{Softmax}\left($	**Converts the raw scores into probabilities.** Each row sums to 1 Gives us the attention weights. (How much this token to every other token.)

Transformers use attention maps to understand relationships.
IVTP needs those attention maps to prune intelligently / w/ logic.

(The math behind token importance scoring)

Eq. 3 – Attention Rollout *(Slide 11)*

Uses Attention Rollout during **Stage 1 (visual token pruning / GTP).**

Groups 3 layers in the ViT.
Computes rolled-out attention map across those layers.
Shows direct / indirect attention paths.

A^l	**The attention map for layer l** Which tokens attend to which tokens in that layer.
I	**Makes sure every token also attends to itself.** - Keeps info from dispearing when layers are multiplied.
$\cdot\, l_1$	**Starting layer in the group** Rollout begins with initial attention map.
$\tilde{A}^l = (A^l + I) \cdot \tilde{A}^{l-1}$	Combine the current layer with The cumulative attention from previous layers.
$l_1 < l \leq l_2$	**Starting and ending layers**

CLS doesn't directly attend to A,
but A is still important through B.

Attention Rollout multiplies attention across several layers (with identity added) to capture direct and indirect relationships between tokens.

Eq. 4 – Importance Score *(Slide 12)*

Calculates how important each visual token is. (General overall)

Used to determine which patches to keep and which to prune in **Stage 1 (Visual Token Pruning / GTP**). (After Attention Rollout)

$\tilde{A}_{cls,i}$	**The rolled-out attention from CLS to token i** *(How much does the CLS token think this patch matters.)* If CLS attends it's probably important.
$\|V_i\|$	**Magnitude (strength) of the patches feature vectors** How strong or informative is this patch on its own. *(Clock face = large magnitude; wall = low magnitude)*
$\sum_j \tilde{A}_{cls,j}\|V_j\|$	**Denominator** Used to normalize. (All importance scores sum to 1)
S_i	**Importance Score for i** Used to prune or keep.

It multiplies how much the CLS token attends to the patch with how strong the patch's visual features are. Then it normalizes across all patches.

High importance scores are kept, and low scores are pruned.

Eq. 5 – CLIP's Text Encoder *(Slide 13)*

This equation shows h ow the instruction/prompt is converted into CLIP vision space so that it can compare text tokens with visual tokens.

Used in Instruction guided pruning (Pruning #2).

1. Questions text is tokenized.
2. Text tokens are passed through CLIP's text encoder.
3. The output embeddings are used to check which parts of the image relate to the question.
4. Tokes irrelevant to the question are removed

T	**The input Text (tokenized)** ("how", "many", "clocks", "are", …)
$\mathcal{T}(T)$	**CLIPs Text Encoder function** Converts text tokens into same space as CLIP's visual embeddings. *Makes it so we can compare text and image.*
$\bar{x}_i^L$	**The vector for the I'th encoded text embedding** Each vector represents the meaning of one word.
S'	**The number of text tokens after e encoding.**
$\mathbb{R}^{S' \times d}$	**Matrix of text embeddings** S' rows = one for each text tokens D columns = embedding dimension *Makes text embeddings compatible with visual embeddings.*

CLIP's text encoder turns the question into vectors that match the image feature space. IVTP uses these text embeddings for instruction-guided pruning.

Eq. 6 – Cosine Similarity *(Slide 14)*

This equation computes how related each text token is to the entire Image.

Measures the similarity between the:

- Visual CLS token
- The I-th text token from the CLIP's encoder

Used in **Stage 2 (Instruction Guided Pruning).**

x_{cls}^{V}	**The visual CLS token (from visual encoder)** - Overall meaning of the image.
$\bar{x}_i^L$	**The I-th text token embedding from the CLIPs Text Encoder** (Becomes a Vector)
$x_{\text{cls}}^{V} \cdot \bar{x}_i^L$	**Dot Product** Measuring the meaning High = Word matches image Low = Word irrelevant
$\|x_{\text{cls}}^{V}\| \ \|\bar{x}_i^L\|$	**Magnitudes / lengths of each vector.** Normalizes so scores are between -1 and 1.
c_i	**The final relevance score for text toke i.** High ci = word relates strongly to image Low ci = word is irrelevant / noisy

Cosine similarity measures how closely each text token matches the visual content, allowing IVTP to keep only the instruction words that matter for pruning in the LLM.

Eq. 7 – Align CLIP Tokens to LLM Tokens *(Slide 15)*

This equation aligns the CLIP Tokens with the LLM text tokens. So that the cosine similarity scores from Equation #6 can be used by the LLM.

Used in **Stage 2 (Instruction Guided Pruning).**

1. Run this Equation
2. Use the aligned scores to filter relevant/irrelevant LLM Tokens.

Without this, CLIP and LLM wont match up.

c_k	**Cosine similarity score from Equation 6**
$T_{\text{CLIP}}(j), \ldots, T_{\text{CLIP}}(j+K)$	**CLIP text tokens that correspond to a single LLM token.** LLM text token = "hydrant" CLIP tokens: - "hy" - "dra"

	- "nt" LLM token maps to 3 CLIP tokens > K = 2
$T_{\text{LLM}}(i)$	**The i-th LLM token** One similarity score per LLM token.
Sum: $\sum_{k=j}^{j+K} c_k$	**Takes all similarity scores of CLIP tokens that belong to the LLM token and adds them.**
$\frac{1}{K+1}$	**Divides by number of tokens.** Computes the mean similarity. (Smooth similarity score for the LLM token.)
C_i	**Final relevance score for the i-th LLM token.**

This equation averages the cosine similarity scores of the CLIP tokens belonging to a single LLM token. (1 score for 1 LLM token.)

Eq. 8 – Pseudo CLS Token *(Slide 16)*

Used in **Stage 2 (Instruction Guided Pruning). (before importance scores.)**

x_{cls}^{V}	**The visual CLS token that represents the whole image.** Only used if no text token is judged relevant.
x_i^L	**The embedding of the i-th text token.** (From LLM Tokenizer)
$I_{\{c_i \geq \tau\}}$	**Indicator function (based on a threshold)** 1 = token relvant 0 = token irrelevant (*Determines if relvant or not)*
c_i	**Smoothed cosine similarity score (from equation #7)**
τ	**Relevance Threshold**
$\frac{1}{\sum I_{\{c_i \geq \tau\}}} \sum x_i^L I_{\{c_i \geq \tau\}}$	**Summation** Unfinished note.

Tables

Table 1 – Pruning Method Benchmarks (Vicuna-7B) *(Slide 18)*

Using Vicuna-7B as the LLM backbone

LLaVA = Baseline (No pruning method.) – [Large Language & Vision Assistant; the combo of a Vision Encoder, Language Model, and Projection Layer]

- Reported by LLaVa authors in their paper.
- Replicated by IVTP authors.

Compares visual token pruning methods on 12 different multimodal benchmarks.

- Accuracy of each method
- Average accuracy across all datasets
- Compute cost (TFLOPs)
- Accuracy Drop & Compute Savings (Compared to full model)
    - LLaVa-1.5-7B

Dataset	What it tests	Example
VQAv2	Visual question answering	**Image**: A man riding a bicycle. **Question**: What is the man riding? **Answer**: A bicycle.
GQA	Complex compositional reasoning.	**Image**: A table with fruit next to a glass. **Question**: What color is the fruit that is next to the glass? **Answer**: Red.
VisWiz	VQA for visually impaired images.	**Image**: A blurry photo of a medicine bottle taken by a blind user. **Question**: What does this label say? **Answer**: Ibuprofen.
SQA	Science question answering. (visual reasoning)	**Image**: Diagram of the water cycle. **Question**: Which process turns liquid water into vapor? **Answer**: Evaporation.
VQAT	Text-heavy VQA. (OCR, reading text)	**Image**: A cereal box showing "Kellogg's". **Question**: What is the brand name on the box? **Answer**: Kellogg's.
POPE	Object hallucination.	**Image**: An empty road with no animals. **Question**: Is there a dog in the image? **Answer**: No.

MME	Multimodal evaluation suite.	**Image**: A bar chart with labeled bars A, B, and C. **Question**: Which bar is the tallest? **Answer**: Bar C.
MMB / MMBCN	Multimodal benchmarks (global + Chinese variants)	**Image**: A dog standing on a skateboard. **Question**: What is the dog standing on? **Answer**: A skateboard.
SEED	Generative comprehension reasoning.	**Image**: Two people sitting at a park bench. **Question**: Describe what the people are doing and why. **Answer**: They are talking and relaxing because they seem to be enjoying the park.
LLaVA-W	World knowledge VQA.	**Image**: A photo of the Eiffel Tower. **Question**: What city is this structure located in? **Answer**: Paris.
MM-Vet	Challenging reasoning / hallucination.	**Image**: A raccoon holding a donut in a kitchen. **Question**: How many food items are visible, and what is the animal holding? **Answer**: Two food items are visible, and the raccoon is holding a donut.

Avg. = Average across all 12 tasks.
TFLOPs = Compute required (Lower is better.)

Summary:

- IVTP keeps almost the full-model accuracy (-1.0%) while cutting compute nearly in half (-46.8%), outperforming all other token pruning methods.

Table 2 – Pruning Method Benchmarks (Vicuna-13B) *(Slide 19)*

LLaVA = Baseline (No pruning method.) – [Large Language & Vision Assistant; the combo of a Vision Encoder, Language Model, and Projection Layer]

- Reported by LLaVa authors in their paper.
- Replicated by IVTP authors.

Compares visual token pruning methods on 12 different multimodal benchmarks.

- Accuracy of each method
- Average accuracy across all datasets
- Compute cost (TFLOPs)
- Accuracy Drop & Compute Savings (Compared to full model)
    - LLaVa-1.5-13B

Avg. = Average across all 12 tasks.
TFLOPs = Compute required (Lower is better.)

Summary:

- IVTP keeps almost the full-model accuracy (-0.8%) the best out of the other token pruning methods shown in the table. While cutting compute nearly in half (-46.9%), outperforming all other token pruning methods.

Table 3 – TFLOPs based on # of Text Tokens *(Slide 20)*

Shows how TFLOPs (compute cost) differs by pruning method as you increase text token count from 128 to 1024.

Again, LLaVA-1.5-7B is the baseline with no pruning method applied.

128 text tokens being short instruction/prompt.
1024 text tokens being long instruction/prompt.

Shows TFLOPs used; % compute saved relative to the full model. (LLaVA)

This is where we can see the slight overhead of IVTP but its so close and unnotivable that its worth the accuracy boost.

Summary:

IVTP: Instruction-based Visual Token Pruning

- Has almost identical compute to other pruning methods.
- Practically identical compute.
- Computation saved matched expected patterns from other methods.
    - Higher savings on shorter text.
    - Lower savings on longer text. (Text = cost)

Table 4 – Dif. Ways of Calculating Attention Scores *(Slide 21)*

An ablation study comparing 3 different ways of computing attention scores.

- Group-wise attention (no rollout)
- Layer-wise attention (no rollout)
- Attention Rollout (IVTP method)

All using the same compute (8.2 TFLOPs), but accuracy differs.

Group-wise (Ablation)	Averages attention across into a big group. (Very noisy, different layers disagree, averaging loses structure.) [Naïve averaging.]
Layer-wise	Importance computed for EACH layer separately. (Extremely noisy, each transformer layer focuses on different tokens.) [More Raw info. then Group-wise] Early = Edges / Colors Mid = Shapes Late = Semantics
Rollout (Group-wise Attention Rollout) (OURS)	Combines layers using attention rollout. (Groups of 3; then prunes based on importance scores.)

Avg. Acc (PI) = Post Instruction Accuracy [Check Pruning Quality]
- How well it performs right after pruning, before full LLM reasoning.

Avg Acc = Average Accuracy
- After full LLM reasoning.

Blue part is compared to baseline LLaVA (not shown other than parenthesis)

Summary:
- Rollout is the best because it uses attention from multiple layers together, which makes the importance scores way more stable and accurate for pruning.

Table 5 – Dif. Ways of Combining Attention Weights *(Slide 22)*

How we merge (aggregate) attention a cross layers.

1. Mean
2. Max
3. Multiply
4. Residual (Ours)

Blue part is compared to baseline LLaVA (not shown other than parenthesis)

Mean	Averages attention across layers. - Accuracy drops a lot. - Loses it structure.
Max	Takes the biggest attention value across layers. - Very noisy. - Overreacts to spikes in attention.
Multiply	Multiples attention without identity/residual.

	- Better result than mean/max. - Sensitive to small variations.
Residual	Attention Rollout (IVTP) - Multiplies Attention but after adding the original information back in. (+ I) - Avoids vanishing attention - Avoids noisy attention - Keeps Important Tokens

Summary:

- Residual rollout works best because it keeps each token's original information (Identity) while mixing in attention from the next layers.

Table 6 – Which Instruction-guided Components Matter *(Slide 23)*

An ablation study testing which parts of the instruction-guided pruning actually matter or are important.

3 Components

1. OTT - Original Text Tokens (LLM question tokens)
2. TE – CLIP Text Encoder (text embedded into CLIP space)
3. RT – Relevance Threshold (filters out weakly related tokens)

Avg. Acc (PI) = Post Instruction Accuracy [Check Pruning Quality]

- How well it performs right after pruning, before full LLM reasoning.

Avg Acc = Average Accuracy

- After full LLM reasoning.

Results:

1. None used = vision only pruning; no question info, no CLIP text, no filtering. (Weak results.) – *Pruning is blind.*
2. OTT only = uses question only. (Weak results) - Doesn't align question to image patches (CLIP encoder); has no threshold so it has noisy important.
3. OTT + RT = uses question and thresholding. F ilters wrong words; important words get removed as there's no way to match question meaning to image patches. (Weak results)
4. OTT + TE + RT = *(Our IVTP has all a.k.a. this one!)*
    a. Uses full question
    b. Aligns with visual tokens (CLIP Text Encoder)
    c. Filters weak irellevant words (Tresholding)
    d. Gives best token importance

    (Good Results)

Summary:

- Table 6 shows that IVTP works best when all three instruction-guided components; the question text, its CLIP embedding, and a relevance

threshold. It gives the highest accuracy and the smallest drop from the full model.

Table 7 – Does LLM-sided Pruning Help Models *(Slide 24)*

The table compares pruning methods and what happens when you apply it also inside the LLM. (Not just the Visual encoder.)

Avg. Acc (PI) = Post Instruction Accuracy [Check Pruning Quality]
- How well it performs right after pruning, before full LLM reasoning.

Avg Acc = Average Accuracy
- After full LLM reasoning.

TopK	Regular TopK Pruning Method
TopK+	TopK w/ LLM-sided Pruning
ToMe	Regular ToMe Pruning Method
ToMe+	ToMe w/ LLM-sided Pruning
IVTP-V	Only vision-side
IVTP	Both vision + LLM-sided Pruning

W/ LLM-sided Pruning:
- TopK accuracy dropped.
  - *Avg. Acc (PI) increases because it removes noisy / irrelevant tokens. Making it look cleaner. (Input for LLM is less cluttered and attention is more focused. But the real important tokens are already gone for Acg Acc.)*
- ToMe accuracy increased with the tiniest bit of improvement.
  - *Avg. Acc (PI) increases because it removes noisy / irrelevant tokens. Making it look cleaner. (Input for LLM is less cluttered and attention is more focused. But the real important tokens are already gone for Acg Acc.)*
- IVTP accuracy increased. (Had higher accuracy even when just vision sided.)

Adding LLM pruning does NOT improve older methods, but it DOES improve IVTP because its first pruning stage is already stable and instruction-aware.

Table 8 – Changing # of Layers Per Pruning Group *(Slide 25)*

How changing the number of layers per pruning group affects accuracy. (In the Group-wise Token Pruning GTP)

Before I said we were using 3 transformer layers for the attention rollout. Well now its time to understand why we chose 3 and not ALL, 2, 3, 4, or 6.

Each Group:
- Computes attention rollout
- Produces importance scores
- Prunes tokens

- Then moves on to the next group

Goal:

What group size gives the best accuracy?

Avg. Acc (PI) = Post Instruction Accuracy [Check Pruning Quality]

- How well it performs right after pruning, before full LLM reasoning.

Avg Acc = Average Accuracy

- After full LLM reasoning.

ALL	• Too many layers merged at once • Attention becomes noisy and unstable • Important signals get averaged out • Poor importance scores → bad pruning
2	• Smaller groups = cleaner attention • Less noise than "ALL" • But still not fully stable
3	• Perfect balance • Enough layers to capture meaningful multi-layer attention flow • But not so many layers that the signal becomes noisy • Produces the most stable and accurate importance scores • Best pruning decisions • Smallest accuracy drop (–1.0%) **SUCCESSFULLY CAPTURES MULTI-LAYER ATTENTION! (NOISE PREVENTED)**
4	• Slightly too large • Attention signal starts getting messy • Still decent, but not as clean as 3 layers
6	• Too many layers • Causes attention wash-out • Important signals drowned • Unstable importance scores → worse pruning

You cannot cut layers because a transformer is built from whole layers but you can do mixed group sizes like some 3, some 4. (Possibly skewing results but not repeatable.)

- *Maybe later layers benefit from slightly more context.*

Summary:

- Table 8 shows that using 3 layers per pruning group gives the best results. Groups that are too small don't capture enough attention flow, and groups that are too big mix too much attention and create noise. Three layers is the perfect balance, giving the highest accuracy and the smallest drop from the full model.

Table 9 – Computational Complexity Comparison *(Slide 26)*

Breaks down the compute cost (TFLOPs) into four components:

- ViT = Computation in Vision Encoder
- LLM = Computation in the Language Model
- Extra = Computation on any Additional Modules the method requires.
- Total = (Self Explanatory)

LLaVa-1.5-7B = Baseline

- No Pruning = Maximum Compute
- How expensive the model normally is.

TopK

- Aggressive pruning visual tokens.
- Thereby reducing amount of work the LLM has to do / reason for.
- Cuts compute almost 50%. (Lower Accuracy as we saw before.)

ToME

- ToMe merges token inside ViT instead of pruning.
- Thereby reducing amount of work the LLM has to do / reason for.

Qwen-VL

- Uses some sort of compression method.
- More expensive in the Visual Encoder.
- Cheaper t han full LLaVA.

IVTP

- GTP Pruning #1 reduces ViT compute.
- GTP Pruning #2 (Instruction-guided) reduces LLM compute.
- Has some extra compute for:
    - Comparing instruction/prompt with visual tokens.
    - Using CLIP text encoder.
    - Using relevant threshold.
- Cuts compute about 47%. (Maintains high accuracy.)