Graduation Thesis

School of Engineering, Tohoku University

Guidance from Keyword for Identifying Small Details in High-Resolution Images with Vision-Language Models

Department of Mechanical and Aerospace Engineering

Image Analysis Laboratory

C0TB1706 Farros Alferro

September 2024

TOHOKU UNIVERSITY

Abstract

School of Engineering
Department of Mechanical and Aerospace Engineering

Bachelor of Engineering

Guidance from Keyword for Identifying Small Details in High-Resolution Images with Vision-Language Models

by Farros Alferro

Humans instinctively zoom in and navigate through high-resolution images to identify small details. However, this intuitive process poses a challenge for current Vision-Language Models (VLMs). These models utilize pre-trained vision encoders that compress images to their native resolution, typically around 336 or 384 pixels, resulting in significant loss of information, particularly for small objects and details. To address this issue, this study proposes a plug-and-play method that utilizes existing model weights without external tools or additional training. By harnessing the combined capabilities of pre-trained vision and text encoders, our method selectively focuses on the most relevant parts of an image, guided by keywords from the query prompt. Our findings indicate that with the right combination of keyword prompting, positional information, cropping techniques, and vision encoder, our approach significantly improves performance over the borrowed model. Moreover, it outperforms both closed-source models, such as GPT4-V and Gemini Pro, and open-source models trained on high-resolution images, including Idefics2 and Dragonfly. Despite these advances, further exploration and experimentation are required to surpass the current state-of-the-art models.

Key Words: computer vision, vision and language, large language model, high resolution image,

Corresponding e-mail address: farros.alferro.t3@dc.tohoku.ac.jp

Contents

1 Introduction **1**
1.1 Problem 1

2 Vision Language Model **2**
2.1 Input Pre-processing 2
2.2 Vision Encoder 4
2.3 Alignment Module 6
2.4 Large Language Model (LLM) 7
2.5 Training Recipes 8

3 Proposed Methods **10**
3.1 Cropping Method 10
3.2 Keyword Extractor Module 12
3.3 Patch Selector Module 14

4 Experiments and Results **18**
4.1 Experimental Setup 18
4.1.1 Mantis 18
4.1.2 V* Bench 18
4.1.3 Multi-V* Bench 19
4.1.4 Implementation Details 20
4.2 Quantitative Results 20
4.2.1 V* Bench 20
4.2.2 Multi-V* Bench 22
4.3 Qualitative Results 23
4.3.1 V* Bench 23
4.3.2 Multi-V* Bench 24
4.4 Ablation Study - V* Bench 24
4.4.1 Vision Encoder 25
4.4.2 Cropping Method 25
4.4.3 Positional Information 26
4.4.4 Keyword Criteria 27
4.4.5 Best Configuration 28
4.5 Ablation Study - Multi-V* Bench 28
4.5.1 Vision Encoder 28
4.5.2 Cropping Method 29
4.5.3 Positional Information 30
4.5.4 Keyword Criteria 32
4.5.5 Best Configuration 33

5 Conclusion **34**

References **38**

1 Introduction

1.1 Problem

Humans naturally recognize on small details in high-resolution images by zooming into specific areas, a capability that poses a considerable challenge for current Vision Language Models (VLMs) [14, 22, 26, 28, 30, 31, 58]. These models typically rely on pre-trained vision encoders like the CLIP [40] image encoder, which processes images at fixed and low resolutions such as 224×224 or 336×336 pixels. Consequently, the models must resize and crop images to fit these resolutions, leading to substantial loss of information. This loss is particularly important when attempting to identify small details in high-resolution images, as crucial visual data can be omitted during resizing and cropping.

Previous research has employed visual search algorithms [49] supported by external detector models such as SAM [24], OwlViT [35], or specially designed tools [20, 47, 48] for visual chain of thought processes. These approaches are effective in addressing the issue, but they demand additional memory due to the external tools and heavily depend on the Large Language Model's (LLM) [4, 13, 41, 55, 56] ability to leverage these tools. Moreover, these methods require training the Vision-Language Model (VLM) from scratch and involve repeatedly passing the sample through the Large Language Model (LLM) in an iterative manner. This can lead to substantial GPU memory usage and extensive inference time.

We propose a plug-and-play solution to this problem that leverages the existing weights of a model without requiring further training. Instead of depending on external models, we utilize the innate capabilities of a pre-trained vision encoder to effectively focus on the most relevant portions of an image, guided by keywords derived from the query prompt. Our results show that through a correct combination of keyword prompting, positional data, cropping techniques, and an enhanced vision encoder, our method greatly surpasses both the adapted model and other current VLMs. This includes both proprietary models such as GPT4-V [36] and Gemini Pro [16], and open-source models like Idefics2 [25] and Dragonfly [12], which are trained on high-resolution images. Despite these improvements, ongoing research and experimentation are required to surpass the performance of the current state-of-the-art model [49].

2 Vision Language Model

Vision language models (VLMs) are multimodal models that learn from both images and text. These generative models receive inputs in the form of images and text and produce text-based outputs. Large vision language models are known for their excellent zero-shot capabilities, meaning they can generalize well across various types of images, such as documents, web pages, and more, without needing specific training on those types. They are used in a range of applications, including discussing images, recognizing images based on instructions, visual question answering, understanding documents, and captioning images. Additionally, some of these models can understand and represent spatial aspects of images. They can generate bounding boxes or segmentation masks to identify or segment specific subjects in images, or even localize and provide information about the positions of different entities. There is significant variety among large vision language models in terms of the data they were trained on, how they process images, and their overall capabilities.

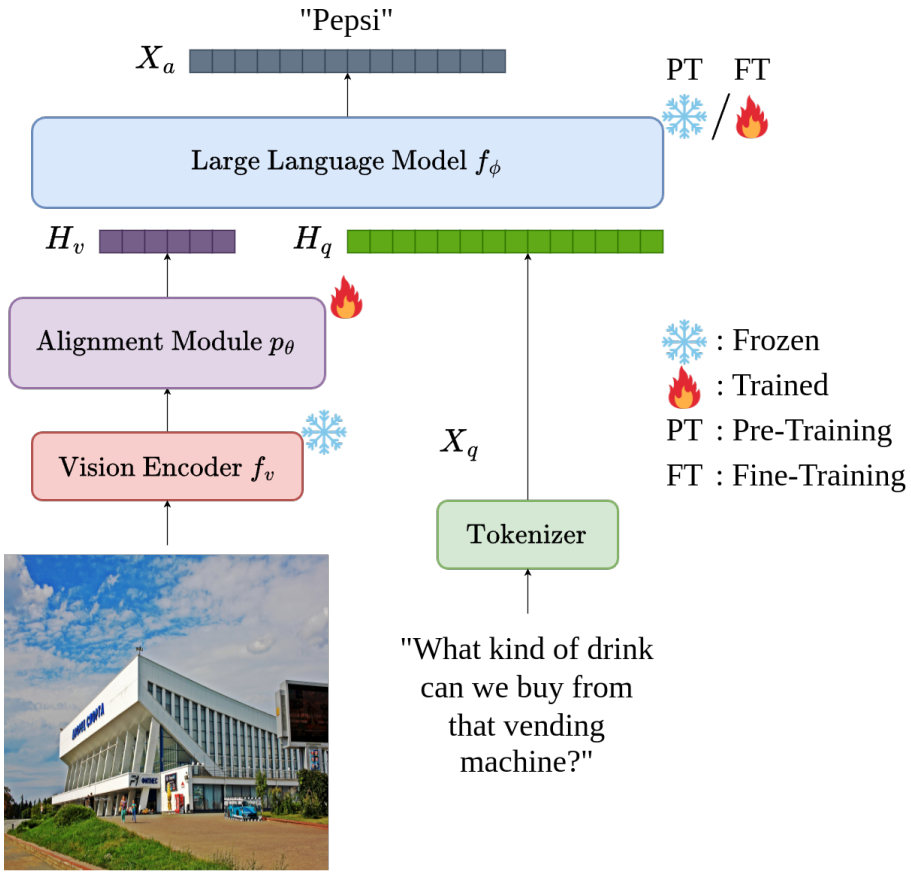


Figure 1: Model overview and training procedures.

2.1 Input Pre-processing

Since a VLM processes both images and text, each type of input is preprocessed separately. For text, it undergoes "tokenization" where a tokenizer breaks down sentences or paragraphs into individual component or "tokens." These tokens act as the fundamental units of language, making it easier for computers to comprehend and process human language by dividing it into

smaller, manageable pieces (see figure 2).

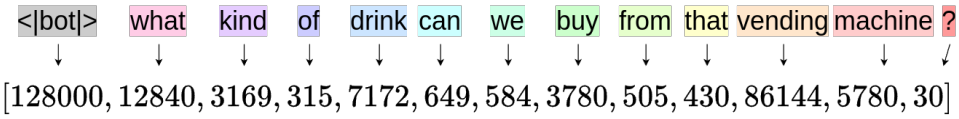


Figure 2: Visualization of how tokenization works. The token "⟨|bot|⟩" is the abbreviation of "⟨|begin_of_text|⟩" token.

The vocabulary size of a large language model (LLM), which serves as a lookup table mapping text to tokens, varies with the model. For instance, the tokenizer from Llama-3 [4] features a vocabulary of 128,256 words. A larger vocabulary size enables the model to encode information more efficiently, potentially enhancing its multilingual capabilities. However, this advantage comes with a tradeoff: larger vocabularies result in bigger embedding input and output matrices, which significantly increase the model's total number of parameters. Additionally, it is important to note that the LLM's responses are decoded from tokens back to text using the inverse of the tokenization process.

Note that the input prompt will not be immediately tokenized, it will first be inserted through a "user-assistant" template so that the LLM can understand the structure of the prompt.

```
<|start_header_id|>system<|end_header_id|>

A chat between a curious user and an artificial intelligence assistant. The
    assistant gives helpful, detailed, and polite answers to the user's
    questions.<|eot_id|><|start_header_id|>user<|end_header_id|>

what kind of drink can we buy from that vending machine?<|eot_id|><|
    start_header_id|>assistant<|end_header_id|>
```

Listing 1: Prompt template

In the text, parts enclosed within ⟨||⟩ are recognized as special tokens. These tokens highlight critical portions of the text, such as "⟨|begin_of_text|⟩", which signifies the start of every prompt. In VLMs, a special token "⟨|image|⟩" is used to denote the position of image tokens, marking where the image-related data begins in the input sequence.

Nevertheless, we won't directly feed these tokens X_q to the LLM. Instead, these tokens serve as indices for a lookup table that stores embeddings from a fixed dictionary. With a vocabulary size of 128,256 and a text embedding dimension of d_q, the embedding matrix will be sized $(128256, d_q)$, where each row represents the vector embedding of a token. These vectors, denoted as $H_q \in \mathbb{R}^{N_q \times d_q}$ where N_q is the number of text tokens, represent the text and will be passed to the LLM. Although this embedding layer is technically part of the LLM, we discuss it separately here to simplify the explanation.

On the other hand, image preprocessing involves three key steps: 1) rescaling the image values to a range between 0 and 1, 2) normalizing by subtracting the mean and dividing by the standard deviation, and 3) resizing the image to the native resolution of the image encoder. The target resolution for resizing can vary depending on the specific vision encoder used. For example, in the case of SigLIP [53] (as shown in figure 3), the resolution may differ from that used by CLIP [40], which resizes images to 336 pixels.

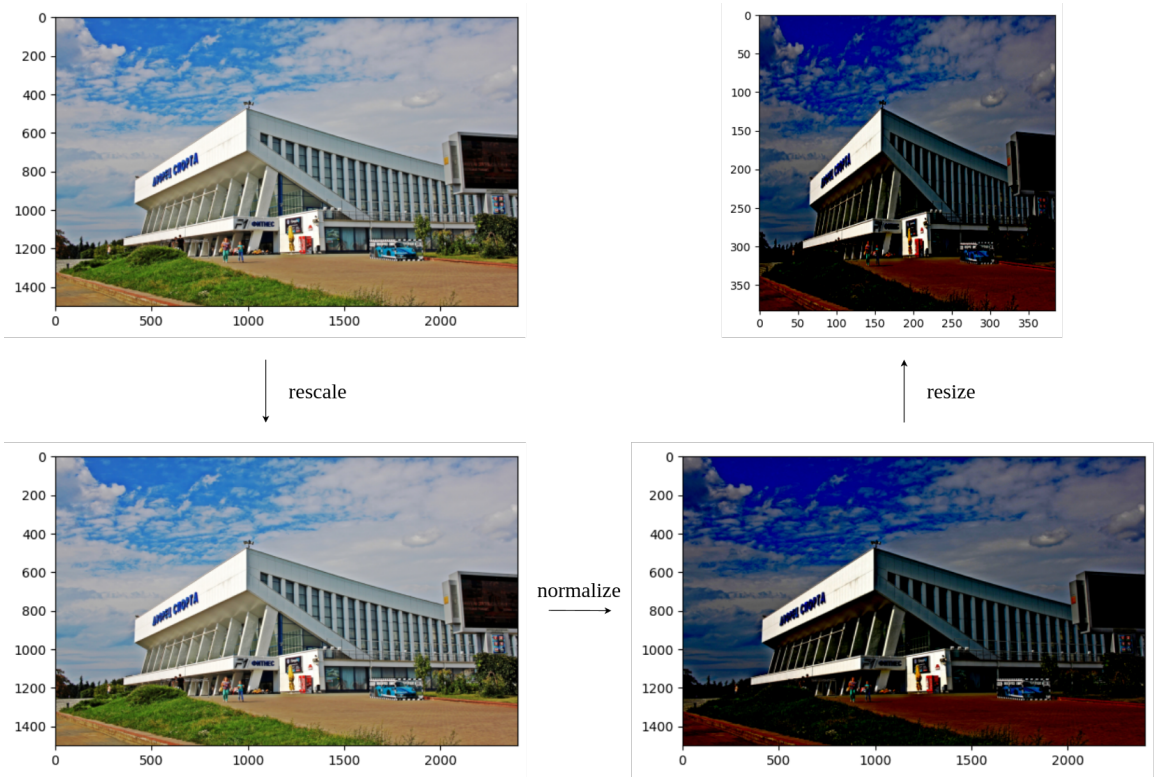


Figure 3: How image is preprocessed. The image is resized from $(2400, 1500)$ to $(384, 384)$.

Once the text and image are preprocessed, the image is sent to the vision encoder and then inserted into the prompt at the position indicated by the "⟨|image|⟩" token.

2.2 Vision Encoder

We process images using a transformer-based architecture called Vision Transformers (ViT), which consists of stacked transformer encoder layers [17]. Initially, the image is segmented into multiple uniform patches. These patches are then treated as tokens and are linearly embedded. Positional information is also incorporated into these tokens to preserve the spatial relationships among them. These embedded vectors are then passed through a Transformer encoder resulting image tokens $X_v \in \mathbb{R}^{N_v \times d_v}$, where N_v is the number of image tokens and d_v is the image embedding space. For image classification tasks, a learnable "classification token" or "cls" denoted as $\mathbf{c} \in \mathbb{R}^{d_v}$ is usually inserted at the start of the sequence (see figure 4).

In VLMs, using a pre-trained vision encoder such as the image encoder from CLIP (Contrastive Language-Image Pretraining) [40] is more practical than developing one from scratch. CLIP is designed to map visual and textual data into the same embedding space using a Vision Transformer (ViT) as its image encoder f_v and a variant of the original Transformer as its text encoder f_u as illustrated in Figure 5. Using CLIP's image encoder in VLMs is advantageous because its pre-training on the image-text contrast task facilitates the integration of images into the text embedding space, a process further supported by the alignment module (see section 2.3).

At a high level, CLIP transforms each input image $\mathcal{I}$ and text $\mathcal{T}$ into d_c-dimensional vector representations $\mathbf{v}$ and $\mathbf{u}$. Specifically, the image encoder first processes each input image $\mathcal{I}$ into

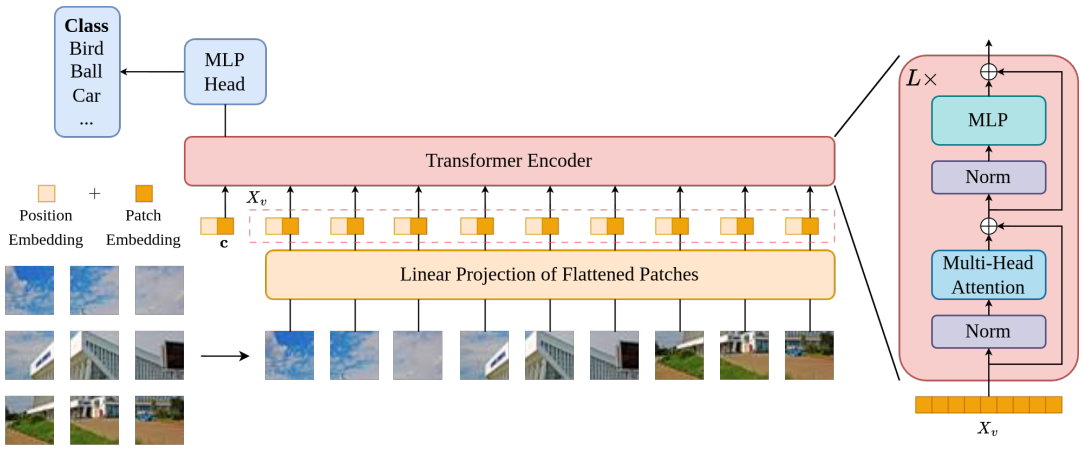


Figure 4: Overview of Vision Transformer.

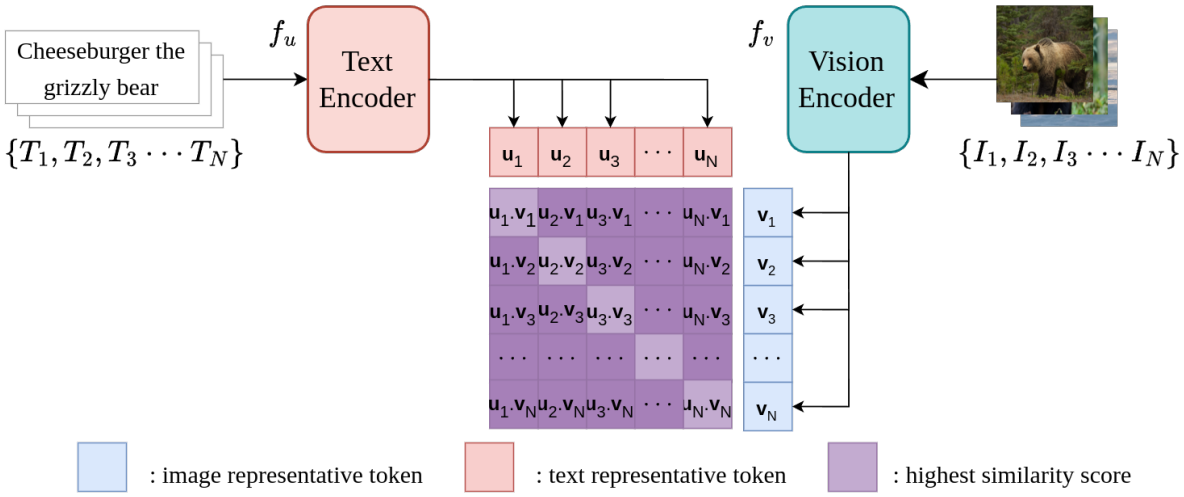


Figure 5: Summary of CLIP approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

image tokens X_v and a "cls" token $\mathbf{c}$. This "cls" token is then transformed by a linear projection g_v to produce the vector $\mathbf{v}$:

$$\mathbf{v} = g_v(f_v(\mathcal{I})[0]) \in \mathbb{R}^{d_c} \tag{1}$$

Similarly, each input text $\mathcal{T}$ is tokenized and input into the text encoder f_u, resulting in text tokens H_u (distinct from H_q). The "end_of_sentence" token, or $\langle|\text{eos}|\rangle$ token $\mathbf{h}_u$, is then passed through a linear projection g_u to generate the vector $\mathbf{u}$:

$$\mathbf{u} = g_u(f_u(\mathcal{T})[\langle|eos|\rangle]) \in \mathbb{R}^{d_c} \tag{2}$$

The projection functions g_v and g_u are designed to map the representations from their respective encoders into a common d_c-dimensional space, facilitating contrastive learning.

During training, with a batch of B image-text pairs $(\mathcal{I}_i, \mathcal{T}_i)_{i=1}^{B}$, CLIP must identify the

correct text for each image from $B \times B$ possible pairings. To achieve this, CLIP develops a multi-modal embedding space by jointly training the image and text encoders to maximize the cosine similarity between $\mathbf{v}$ and $\mathbf{u}$ of the B correct pairs in the batch while minimizing the cosine similarity of the embeddings for the $B^2 - B$ incorrect pairings. The optimization of these similarity scores utilizes symmetric cross entropy loss for both image-to-text and text-to-image contrasts.

$$\mathcal{L}_i^{(v \to u)} = -\log \frac{\exp(\langle \mathbf{v}_i, \mathbf{u}_i \rangle / \tau)}{\Sigma_{k=1}^{B} \exp(\langle \mathbf{v}_i, \mathbf{u}_k \rangle / \tau)} \tag{3}$$

$$\mathcal{L}_i^{(u \to v)} = -\log \frac{\exp(\langle \mathbf{u}_i, \mathbf{v}_i \rangle / \tau)}{\Sigma_{k=1}^{B} \exp(\langle \mathbf{u}_i, \mathbf{v}_k \rangle / \tau)} \tag{4}$$

where $\langle \mathbf{v}, \mathbf{u} \rangle$ denotes the cosine similarity, specifically, $\langle \mathbf{v}, \mathbf{u} \rangle = \mathbf{v}^T \mathbf{u} / \|\mathbf{v}\| \|\mathbf{u}\|$. Additionally, $\tau \in \mathbb{R}^+$ represents a temperature parameter, established as a learnable parameter in the model. The final training loss is computed by averaging these two losses:

$$\mathcal{L} = \frac{1}{B} \sum_{i=1}^{B} \Big(\frac{\mathcal{L}_i^{(v \to u)} + \mathcal{L}_i^{(u \to v)}}{2} \Big) \tag{5}$$

Another work suggests employing a sigmoid-based loss in place of the softmax-based contrastive loss [53]. The sigmoid-based loss evaluates each image-text pair individually, transforming the learning challenge into a standard binary classification task on the dataset comprising all possible pair combinations. It assigns positive labels to the matching pairs $(\mathcal{I}_i, \mathcal{T}_i)$ and negative labels to all mismatched pairs $(\mathcal{I}_i, \mathcal{T}_{j \neq i})$. The loss is defined as follows:

$$\mathcal{L}_{SigLIP} = -\frac{1}{B} \sum_{i=1}^{B} \sum_{j=1}^{B} \log \frac{1}{1 + \exp^{z_{ij}(-\tau \langle \mathbf{u}_i, \mathbf{v}_j \rangle + b)}} \tag{6}$$

where z_{ij} is the label for a given image and text input, equaling 1 if they are correctly paired and -1 otherwise. Additionally, b represents an extra learnable bias term.

In contrast to CLIP, in VLM, we eliminate the "cls" token, retaining only the image tokens $X_v \in \mathbb{R}^{N_v \times d_v}$. These image tokens hold the visual information of the image, which is then conveyed to the alignment module and the large language model for further processing.

2.3 Alignment Module

The alignment module in the Vision Language Model architecture takes the X_v image tokens and projects them into $H_v \in \mathbb{R}^{M_v \times d_q}$, where M_v is the number of the aligned image tokens and d_q is the embedding dimension similar to the text token (see figure 1). This process effectively maps the image features from the image embedding space to the language model's textual embedding space, enabling the image to be interpreted by the language model. The most common form of this module is a Multi-Layer Perceptron (MLP), which typically consists of two linear layers with a GELU activation function [21] between them [31]:

$$H_v = \text{GELU}(X_v Z_v^{(1)}) Z_v^{(2)} \tag{7}$$

where $Z_v^{(1)} \in \mathbb{R}^{d_v \times d_q}$ and $Z_v^{(2)} \in \mathbb{R}^{d_q \times d_q}$ are learnable parameters.

Note that in the case of an MLP, the number of visual tokens is preserved (i.e., $M_v = N_v$). Other more sophisticated methods, such as the resampler [7] and Q-former [28], employ M_v learnable queries (typically $M_v < N_v$ for efficiency) and use cross-attention to extract visual cues from the visual features. Additionally, some works attempt to reduce M_v by implementing a convolution layer [11, 18] or adaptive average pooling [51] to streamline the process.

2.4 Large Language Model (LLM)

Now that we have both the image tokens H_v and prompt tokens H_q with the same embedding dimension d_q, we feed these tokens into the large language model (LLM) f_ϕ to get the answer to our question X_a. In this setup, the image tokens replace the special "⟨image⟩" token. The position of this special token varies depending on the task. For example, in an image captioning task, it is typically placed at the beginning of the prompt. In visual question answering, it is often interleaved within the text. Some models are even designed to handle multiple images within a single prompt, requiring the use of several "⟨image⟩" tokens at various positions.

There is a diverse range of open source LLMs available [1, 2, 4, 5, 27, 34, 50, 56]. These LLMs are primarily built on stacks of the Transformer decoder [45], originally designed to generate sequences from encoded information as shown in Figure 6. In current implementations, the decoder has been modified to manage both the understanding and generation of text in an autoregressive manner, producing text one word at a time and utilizing previous outputs as context. Additionally, modern implementations incorporate advanced techniques like rotary position embedding (RoPE) [43], grouped-query attention [6], root mean square normalization [54], KV-cache [39], and SwiGLU activation function [42] to enhance performance and efficiency while minimizing parameter count.

In the decoder layers of these models, the self-attention mechanism plays a crucial role [9], enabling the model to consider all previous words in the sequence when predicting the next one. This feature is vital for maintaining long-range dependencies within the text. Moreover, in a typical Transformer decoder, self-attention is masked to ensure that predictions at any given position are based solely on earlier, already known outputs. This masking is essential to prevent the model from "seeing the future" during training, thus preserving its autoregressive nature, where each prediction relies sequentially on the previous ones. These models are trained to emulate human-like understanding and generation of text.

To ensure their effectiveness, LLMs are trained on vast text corpora, often encompassing billions of pages. For example, Llama 3 was trained with 15 trillion tokens [4]. This extensive training allows them to develop an understanding of grammar, semantics, and conceptual relationships through techniques such as zero-shot and self-supervised learning. Once fully trained, LLMs can generate text by autonomously predicting subsequent words based on the input provided and their accumulated knowledge. This ability enables them to produce coherent and contextually appropriate text, making them valuable for various tasks in natural language understanding and content generation.

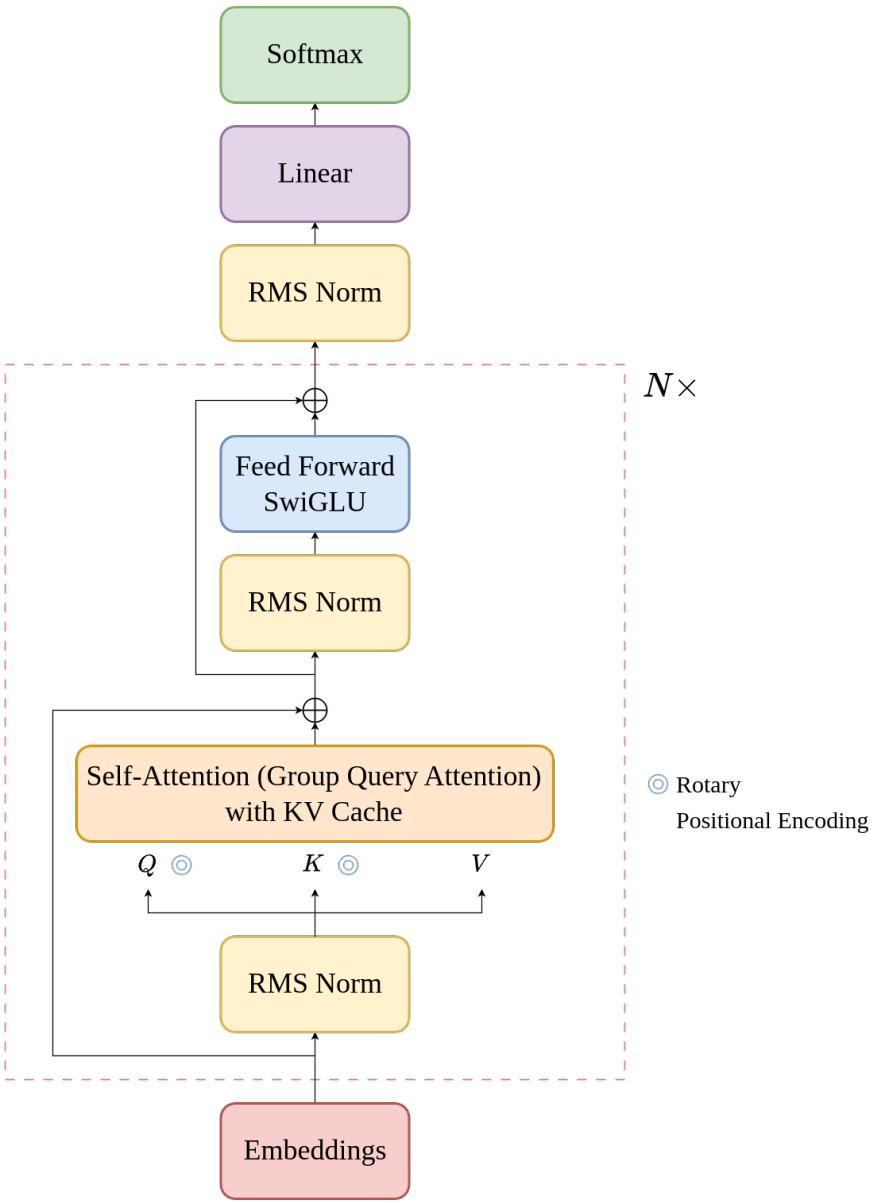


Figure 6: The Llama 3 model architecture [4].

2.5 Training Recipes

In the structure of VLMs, there are three primary components: the vision encoder, the alignment module, and the LLM. Both the vision encoder and LLM are pre-trained on their specific types of data—image-text pairs for the vision encoder and text-only data for the LLM. Although the vision encoder is adept at mapping image features into a text embedding space, it maps them to a space different from that used by the LLM. Furthermore, since the LLM is trained solely on text, it has no capacity to recognize or interpret images directly.

The alignment module is central in VLMs, as it adjusts the dimensional alignment between image tokens from the vision encoder and prompt tokens used by the LLM. This adjustment, discussed in 2.3, allows the LLM to "see" and interpret the input images. The module's ability to accurately map image tokens is honed through a specialized training process. In common, training of VLM typically occurs in two distinct phases (see figure 1):

1. **Pre-training**: This phase is dedicated to align vision and language features. It involves training the initially randomized weights of the alignment module ($Z_v^{(1)}$ and $Z_v^{(2)}$) in isolation, with all other model components frozen. This stage often employs image captioning tasks, which focus solely on aligning the visual and textual data without the complexities of deeper contextual understanding.

2. **Fine-tuning**: During this stage, only the vision encoder is frozen, and the focus shifts to train both the projection module and the LLM to handle more complex and interactive tasks such as visual question answering and interleaved image-text understanding. This stage aims to prepare the LLM to process and understand tasks that integrate both visual and textual elements. Typically, this phase involves training with specialized or proprietary datasets to refine the model's capabilities. For instance, [12] uses medical data for finetuning.

These training stages are crucial for the VLM to perform effectively across various visual-textual tasks.

Other approaches to training include methods that only fine-tune the model [23], strategies that involve training the vision encoder as well [18], techniques that incorporate more than two training phases [12], and practices that focus on training just the last few layers of the vision encoder [57], among others.

3 Proposed Methods

Since the VLMs previously discussed are limited by their vision encoders to processing only relatively low-resolution images, they struggle to identify small details in high-resolution images. To address this limitation, we employ a different image preprocessing method that allows the VLM to perceive the image with high detail. Additionally, we propose the integration of plug-and-play modules. These modules are designed to attach seamlessly to existing VLMs without disrupting their core functionalities. The primary function of these modules is to enable VLMs to process high-resolution images and accurately identify small, specific objects within those images. This enhancement allows VLMs to maintain their performance while extending their applicability to more detailed visual tasks.

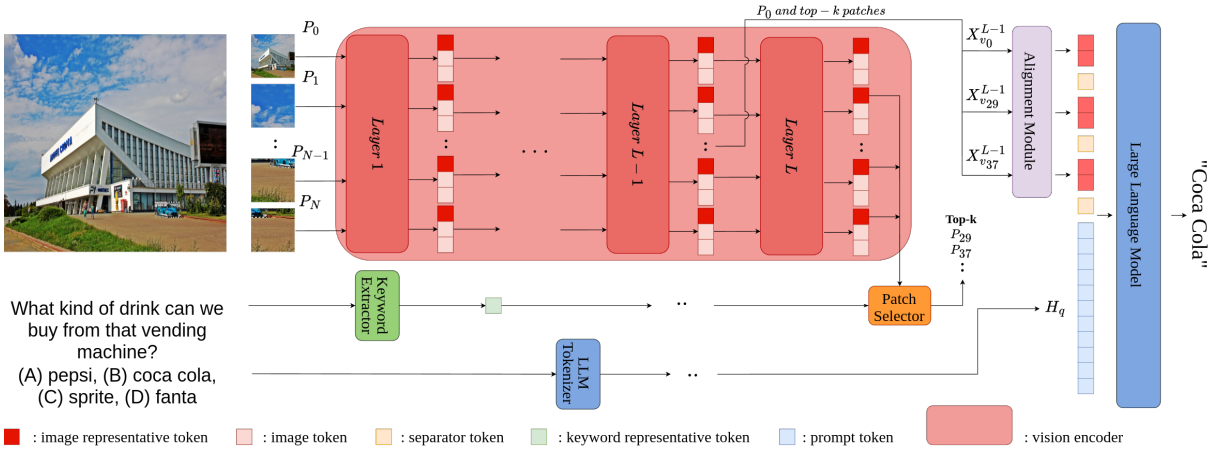


Figure 7: The overall architecture of the proposed model.

3.1 Cropping Method

As discussed in 2.1, typical VLMs resize (and crop) input images to fit their vision encoder's native resolution, a process that often results in the loss of small details, especially in high-resolution images. To minimize this loss of information, we employ a different preprocessing approach: we resize the image $\mathcal{I}'$ so that its dimensions are a multiple of the native resolution (W', H'). Subsequently, we crop the image following the native resolution s, resulting in patches $P \in \mathbb{R}^{N_p}$ where N_p is the number of patches, as illustrated in figure 8. This method ensures that the image not only fits the required input size but also retains more of its original detail by selectively focusing on smaller sections without significantly reducing the overall resolution.

In our approach, we implement two cropping strategies: dynamic and naive. The dynamic strategy begins by calculating the original image's aspect ratio as $\frac{W}{H}$. We then seek the new aspect ratio that minimally deviates from the original by exhaustively searching through potential n_w and n_h values, where n_w and n_h are the number of patches along the x and y axes, respectively. This is expressed in the following equations:

$$(n_w, n_h) = \min_{(1,1) \sim \left(\left\lceil \frac{W}{s} \right\rceil, \left\lceil \frac{H}{s} \right\rceil\right)} \left| \frac{W}{H} - \frac{n_w}{n_h} \right| \tag{8}$$

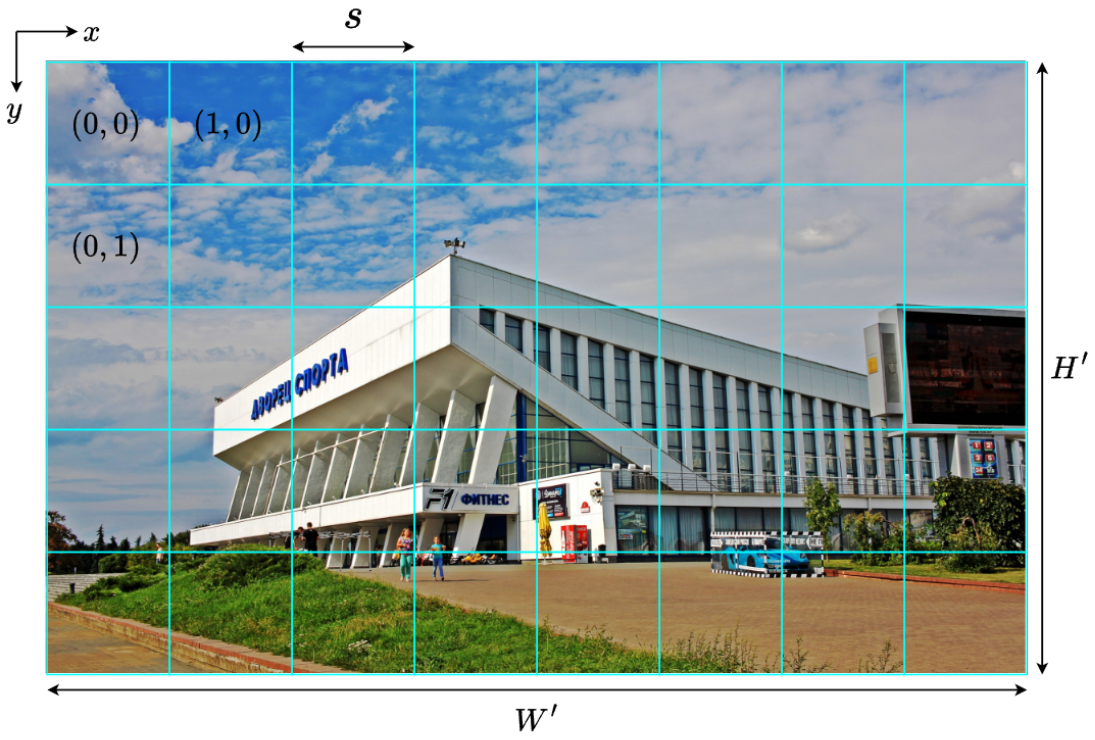


Figure 8: The image is resized from $(W, H) \rightarrow (W', H')$, then cropped according to the native resolution as highlighted by the blue box.

$$W' = W_{dy} = n_w \times s, \;\; H' = H_{dy} = n_h \times s \tag{9}$$

This method aims to preserve the original aspect ratio while resizing the image to match the native resolution. For the naive method, we obtain the number of patches along the x and y axes by naively dividing the original resolution with the native resolution:

$$(n_w, n_h) = \left(\left\lfloor \frac{W}{s} \right\rfloor, \left\lfloor \frac{H}{s} \right\rfloor \right) \tag{10}$$

$$W' = W_{nv} = n_w \times s, \;\; H' = H_{nv} = n_h \times s \tag{11}$$

Each cropping method presents distinct advantages and disadvantages. The dynamic cropping method, which focuses on preserving the original aspect ratio, often results in fewer patches (n_w and n_h) compared to the naive method. This can lead to a compression of the image's information, as fewer patches mean that each patch might contain a more condensed representation of the image, as depicted in Figure 9. Conversely, the naive method does not maintain the aspect ratio, which can distort the image's original proportions, but it typically allows for a clearer and more detailed view of each patch. This method can capture more information per patch, making it useful for detailed analysis, albeit at the potential cost of altering the image's overall appearance.

Among these patches, at least one will contain the queried object, such as the patch at coordinates $(4, 3)$ in Figure 8, which includes the "vending machine." We aim to feed this specific patch to the LLM so it can return the correct answer. However, why not feed all the patches to the LLM? There are three primary reasons against doing so:

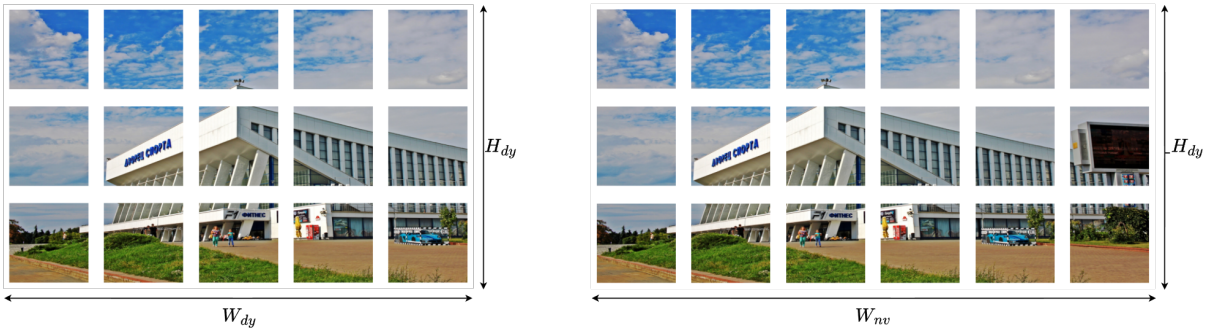


Figure 9: Comparison of the dynamic (left) and naive (right) methods. The dynamic method maintains the aspect ratio but results in lower resolution patches, while the naive method does not keep the aspect ratio but produces clearer patches.

- It's redundant. We want to avoid having the LLM process patches that do not contain the queried object.
- It places a burden on the GPU. Each patch results in a large number of tokens (576 for CLIP and 729 for SigLIP), leading to increased memory consumption.
- The LLM has a limited context length (8k tokens for Llama 3), which restricts the number of tokens it can handle. To avoid exceeding this limit, we carefully select only those patches that contain the queried objects for input.

To select the correct patches, we will first pass all the patches through the vision encoder. Then, we will use the keyword extractor and patch selector modules to identify the relevant patches. Details on these modules will be provided in subsequent sections.

3.2 Keyword Extractor Module

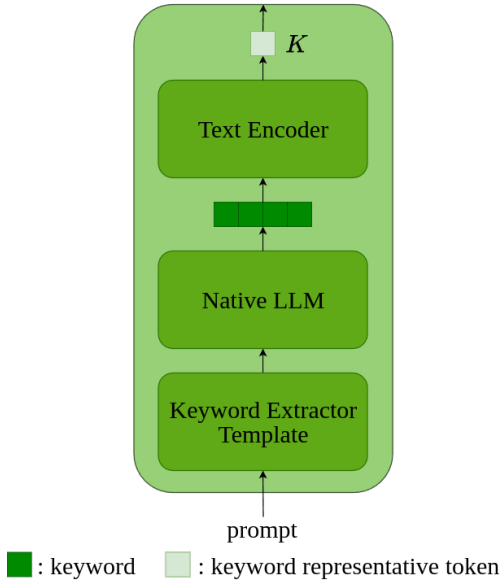


Figure 10: The overview of the keyword extractor module.

First, we will isolate the queried object or the "keyword" from the given prompt. We do this to leverage the inherent capability of the vision encoder and text encoder for image-text matching (refer to Section 2.2). Since the vision encoder and text encoder were pre-trained using relatively simple text (e.g., "Cheeseburger the grizzly bear" or "a photo of a grizzly bear"), inputting the entire prompt could lead to a distribution shift and yield suboptimal results.

To achieve this, we insert the prompt into a few-shot template, where we have already provided several question-and-answer examples. We then feed this template to the native LLM. We utilize the few-shot approach because it guides the LLM to return the answer in the desired format, making it easier to extract the keyword. Below is the template we use:

```
<|start_header_id|>system<|end_header_id|>

Use the text below as an example to generate your answers to the user's query. Give the answer in the same format.

Example starts:
```
User: What is/are the object(s) that being asked in below question? Also give some useful visual features that best describes each object in a photo.
'What kind of drink can we buy from that vending machine?'
Assistant: The object being asked is vending machine. Several visual features of the object are:

'vending machine':

* typically have a large, upright, rectangular shape.
* usually have a large glass or transparent plastic front
* often feature logos, product images, and labels on their exterior
* Most are metallic and have a dominant color (often bright or neutral)
```
Example ends

Example starts:
```
User: What is/are the object(s) that being asked in below question? Also give some useful visual features that best describes each object in a photo.
'Is the wallet on the left or right side of the keyboard?'
Assistant: The objects being asked are wallet and keyboard. Several visual features of the objects are:

'wallet':

* typically have a compact, flat, rectangular shape.
* can be made from various materials including leather, synthetic fabric, or even metal for hard cases.
* generally small enough to fit in a pocket or a small bag.
* come in a wide range of colors, from classic black or brown to vibrant hues and patterns.

'keyboard':

* typically feature a rectangular array of keys in a grid layout.
```
```

```
* can be made from plastic, metal, or other materials.
* come in various colors, although black and white are most common.
* may have a visible USB cable or may be identified as wireless if there is no cable connected.
```
Example ends
Please generate answer in the SAME FORMAT as shown in the above examples. Your response must have an equal number of features for each object in the question.
Please ensure to cover all significant visual features.<|eot_id|><|start_header_id|>user<|end_header_id|>

{prompt}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Listing 2: Few-shot template used to extract the keyword.

Then we take the keyword(s) using regex [3].

Before passing it to the text encoder, we equip the keyword(s) with one of three templates:

- word: Only the keep the keyword "{keyword}". For instance, the word "vending machine".
- template: We add a short fix description to the keyword "a photo of a {keyword}". For instance, "a photo of a vending machine".
- description: We utilize the extracted visual description to form sentences:
  1. "a {keyword}, which {description 1$^{\text{st}}$}"
  2. "a {keyword}, which {description 2$^{\text{nd}}$}"
  3. "a {keyword}, which {description 3$^{\text{rd}}$}"
  4. ...

  For instance, "a vending machine, which typically have a large, upright, rectangular shape".

Afterward, we feed these formatted texts to the text encoder to derive the keyword representative token $K \in \mathbb{R}^{d_c}$ (the $\langle|\text{eos}|\rangle$ token). This text encoder is the text encoder that has been pre-trained with the vision encoder (see Figure 5). In the case of the description template, we process each description separately, resulting in $N_k$ number of keyword representative tokens. For the sake of simplicity, throughout this document, we will use the word template ($N_k = 1$).

## 3.3 Patch Selector Module

Now that we have obtained the keyword representative token $K$, we can use it to identify the relevant patches. This is achieved by calculating the cosine similarity between $K$ and the patch representative tokens $C \in \mathbb{R}^{N_p \times d_c}$, which are derived from the vision encoder. These patch representative tokens correspond to the "cls" token $\mathbf{c}$ for each patch. We then select the top-$k$ patches with the highest cosine similarity scores and store their indices in $S \in \mathbb{R}^k$. The patch selector module is depicted in Figure 11.

The idea behind this method is that since the vision and text encoders were pre-trained on an image-text matching task (as detailed in Section 2.2), the patch with the highest similarity
```

score is most likely to contain the queried object. However, we choose to select the top-k patches to enhance the likelihood of capturing the desired patches. In some instances, the most relevant patch may contain noise due to the presence of other objects, or the queried object may be scattered across multiple patches. This scattering can result in the patch with the queried object not achieving the highest similarity score. By selecting the top k patches, we increase the chances of including all necessary patches.

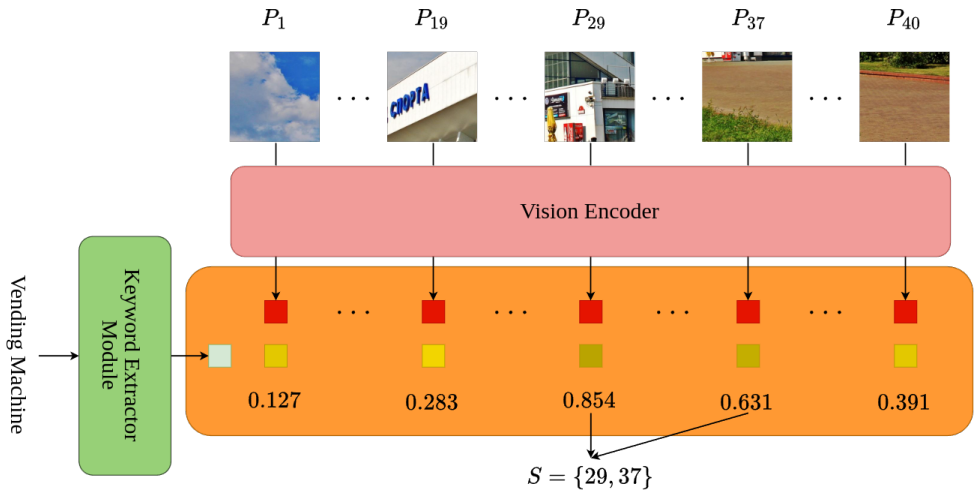


Figure 11: How the patch selector module works. Here we set $k = 2$.

Afterward, we extract the image tokens from the $L-1$ layer of the vision encoder based on the indices S, resulting in the top-k image tokens $X_{v_{topk}}^{L-1} \in \mathbb{R}^{(k+1)\times N_v \times d_v}$. We choose tokens from the $L-1$ layer because the VLM's alignment module we adopted is trained to work with outputs from this layer of the vision encoder. In addition to the top k patches, we include the entire image patch P_0 (which has been cropped and resized), anticipating that the LLM will be able to pinpoint the location of the queried objects within these patches. Including the whole image patch also aids the LLM in understanding the global context of the image. Finally, these image tokens are passed to the alignment module resulting in $H_{v_{topk}}^{L-1} \in \mathbb{R}^{(k+1)\times M_v \times d_q}$.

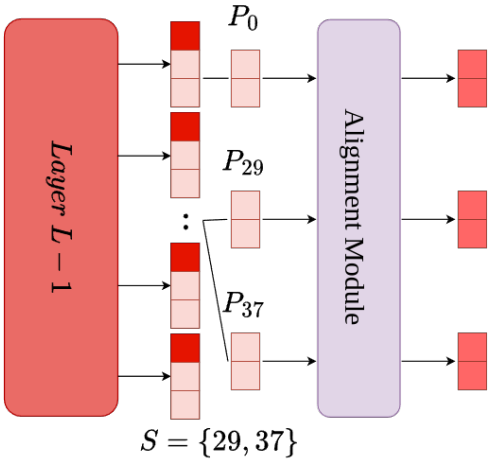


Figure 12: The image tokens are passed to the alignment module for dimension alignment with the prompt token.

Before concatenating the image tokens with the prompt tokens, we add a separator token

between the patches. This token is the embedding representation of the text ",". The inclusion of this separator token assists the LLM in distinguishing between the image patches and the prompt token. Additionally, we add positional information to the patches to help the LLM understand the spatial relationships between them. This is accomplished by incorporating positional encodings into the image tokens. We have adopted three different methods for positional encoding (see Figure 13):

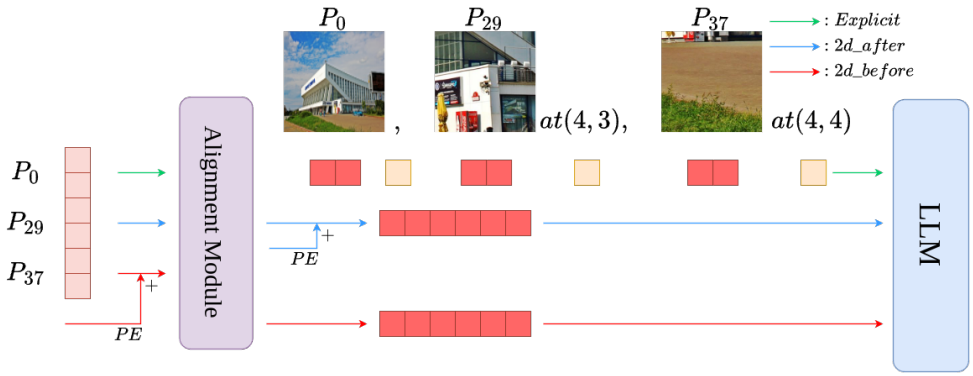


Figure 13: Illustration of the three positional encoding methods.

- Explicit: We explicitly add the coordinates of the patches, as shown in Figure 8, along with the separator token. This method is simple and straightforward, but it may result in longer inference time due to the increased number of tokens.

- 2d_before: We implement the absolute 2D positional encoding, an expansion of the method used in the original transformer [45], to include both dimensions, x and y. The positional encodings for each axis are calculated as follows:

$$
\begin{aligned}
PE_{(x,2i)} &= \sin\left(\frac{x}{10000^{2i/d/2}}\right) \\
PE_{(x,2i+1)} &= \cos\left(\frac{x}{10000^{2i/d/2}}\right) \\
PE_{(y,2i)} &= \sin\left(\frac{y}{10000^{2i/d/2}}\right) \\
PE_{(y,2i+1)} &= \cos\left(\frac{y}{10000^{2i/d/2}}\right)
\end{aligned}
$$

where both PE_x and PE_y dimensions are $\mathbb{R}^{n_t \times d/2}$. We then broadcast and concatenate these encodings along the channel dimension to obtain:

$$
PE_{(x,y)} = [PE_x, PE_y] \in \mathbb{R}^{nt \times d}
$$

with x and y representing the horizontal and vertical coordinates of the patches, respectively, n_t being the number of tokens for each patch, and d being the embedding dimension of each token. This positional encoding is added to the selected patches (excluding the entire image patch) before passing the image tokens to the alignment module, which means $nt = N_v$ and $d = d_v$. This method does not introduce additional tokens, making it more efficient than the explicit method. However, since the alignment module and the LLM

were not trained with this positional encoding, it could potentially affect the model's performance.

- 2d_after: This method is similar to 2d_before, but we add the positional encoding after the alignment module. Here, $nt = M_v$ and $d = d_q$. Like the 2d_before method, this approach does not add extra tokens but may impact the model's performance because the LLM was not trained with this positional encoding.

Finally, we insert the image tokens of the patches (together with the separator tokens and the coordinate tokens in case of explicit method) into the prompt token, replacing the placeholder token $\langle|image|\rangle$ to form the input sequence X_{input}. This input sequence is then passed to the LLM for processing, which will generate the answer to our query.

4 Experiments and Results

4.1 Experimental Setup

4.1.1 Mantis

We chose an open-source VLM named Mantis [22] as our primary model, which offers two vision encoder variants: OpenAI's CLIP [40] and Google's SigLIP [53] (see 2.2 for details). It incorporates an MLP alignment module and uses Llama-3-8B Instruct from Meta AI [4] as the LLM. This model enhances the typical VLM capabilities by supporting multiple image inputs. It was pre-trained on the CC3M 558k image-captioning dataset with a learning rate of $1e-3$, a single epoch, a batch size of 256, a warmup ratio of 0.03, and a cosine learning rate scheduler. For fine-tuning, it utilized the Mantis-Instruct 721k multi-image instruction tuning dataset with the same setup as the pre-training, except the learning rate was reduced to $1e-5$ and the batch size was halved to 128. The fine-tuning process ran on 16 A100 GPUs using DeepSpeed Zero-3 [8] and Flash-attention2 [15].

4.1.2 V* Bench

We evaluated our model using V* Bench [49], which consists of 191 high-resolution visual question and answer tasks sourced from SAM, with an average image resolution of 2246×1582 pixels. V*Bench is organized into two sub-tasks: attribute recognition and spatial relationship reasoning. The attribute recognition task includes 115 samples that challenge the model to identify specific attributes (e.g., color, material) of an object. The spatial relationship reasoning task consists of 76 samples and requires the model to determine the relative spatial positions between two objects. These tasks are designed to assess the model's ability to conduct detailed visual analysis in multimodal contexts.



(a) Attribute task $(1933, 1500)$ px.



(b) Spatial task $(2250, 1500)$ px.

Figure 14: Examples from V* Bench. Left image prompt: "What is the color of the little girl's shirt?" Right image prompt: "Is the blue luggage on the left or right side of the bus?"

4.1.3 Multi-V* Bench



(a) Figure with resolution of $(1920, 2880)$ px. (b) Figure with resolution of $(1920, 2880)$ px.

Figure 15: One example of attribute task from Multi-V* Bench. The prompt: "What is the color difference of the motorcycle between these images?"

Since our plug-and-play modules do not interfere with the built-in abilities of the adopted model, and since Mantis was designed to process multiple images, our model is also capable of identifying small details among multiple images. To support our argument, we assessed our model using our custom benchmark, Multi-V* Bench, which includes multi-image, high-resolution visual question and answer tasks with an average resolution of 1920×2475 pixels sourced from Unsplash [44] and Pexels [38]. The benchmark consists of four tasks, each comprising 20 samples:

- **Attribute**: Identify attributes differences of an object between images. There are 43 images in total with resolution of 1920×2622 pixels.
- **Existence**: Determine if an object exists between the images. There are 47 images in total with resolution of 1920×2419 pixels.
- **Counting**: Count the number of objects among the images. There are 56 images in total with resolution of 1919×2467 pixels.
- **OCR**: Extract or identify text from the images. There are 55 images in total with resolution of 1920×2414 pixels.

The avarage number of images per task is 2.5, with the maximum number of images per task is 4. Furthermore, we enhance the clarity of the crafted question and choice by utilizing GPT4 (`gpt-4o` model) [36] with the following prompt:

```
I'm creating a benchmark that asks about the content of images. It consists
of multiple-choice questions with 2 to 4 answers. I'm going to give you the
question and the answers, and what you need to do is to fix any grammatical
errors and make the question and answers sound more natural. Here is the
question and the answers:

{prompt}

only return the fixed question and answers.
```

Listing 3: Prompt for GPT4 to enhance the question and answers.

Then, small manual adjustments were done to finalize the benchmark.

4.1.4 Implementation Details

We integrated the plug-and-play modules into the official Mantis model and processor using PyTorch [37] and HuggingFace's Transformers [46]. Our custom evaluation pipeline was developed in accordance with the evaluation script from LLaVA [30] and utilized HuggingFace's accelerate package [19], following the original Mantis implementation. However, we opted for scaled dot product attention (SDPA) instead of the Flash-attention2 used during training. The evaluation script was run on our lab's server using the Slurm Workload Manager [52], on a node equipped with three Nvidia RTX A6000 48 GB GDDR6 GPUs, 100 GB of memory, and 16 CPUs per task. It's important to note that all models from Table 1, with the exception of any private models, were evaluated using the same hardware and pipeline.

4.2 Quantitative Results

4.2.1 V* Bench

As shown in Table 1, our models significantly improve baseline performance by substantial margins: 7.29% and 16.76% on the overall score for the CLIP and SigLIP variants, respectively. Specifically, our plug-and-play modules notably enhance the attribute score, which is expected since the LLM can observe the queried object in greater detail. Furthermore, the SigLIP variant's performance doubles that of the original score, likely due to its vision encoder being pre-trained on higher resolution images compared to CLIP's vision encoder. However, the spatial scores show only a minor increase for both variants. This could be due to the positional encoding not effectively conveying the locations of objects to the LLM, despite the enhanced visibility. Additionally, the inference time for our models are significantly slower than those of the baseline models, largely because they process a larger number of tokens—while the baseline models handle just one image, our models process multiple images. The prolonged inference time is primarily attributed to the models requiring two passes through the LLM: one for keyword extraction and another for generating the answer. Further details on the comparison between variants are available in Section 4.4.

Method	Attibute (%)↑	Spatial (%)↑	Overall (%)↑	Inference Time (sample/s)↓	Training	External Models
Human	98.26	100.00	98.95	-	-	-
Random Guess	26.73	50.00	35.99	-	-	-
Open-source models						
LLaVA-1.5 [30]	44.35	52.63	47.64	0.314	✓	×
Mantis-CLIP ‡[22]	35.65	61.84	46.07	**0.257**	✓	×
Mantis-SigLIP ‡[22]	33.04	59.21	43.46	<u>0.295</u>	✓	×
High resolution models						
Mini-Gemini 8B [29]	60.87	63.16	61.78	1.218	✓	×
Dragonfly 8B [12]	49.57	52.63	50.79	2.865	✓	×
Idefics2 8B [25]	54.78	56.58	55.49	1.657	✓	×
LLaVA-NeXT 8B [10]	60.87	61.84	61.26	1.149	✓	×
LLaVA-HR [32]	46.09	52.63	48.69	0.554	✓	×
tool-using models						
SEAL (SOTA) [49]	**77.39**	**77.63**	**77.48**	94.25	✓	✓
Private models						
GPT-4V Preview [36]	42.61	46.05	43.98	-	-	-
GPT-4 Turbo [36]	46.09	53.95	49.21	-	-	-
GPT-4o [36]	<u>69.57</u>	52.63	<u>62.83</u>	-	-	-
GPT-4o Mini [36]	49.57	<u>63.16</u>	54.97	-	-	-
Ours-CLIP	52.17	53.95	52.88	5.268	×	×
Ours-SigLIP	63.48	61.84	<u>62.83</u>	5.479	×	×

Table 1: Evaluation of vision language models on V* Bench [49]. Best results are in bold, and the second-best performances are underlined. ‡ indicates our base model with their respective vision encoder.

We also compared our models with the GPT-4 family and other available open-source models, particularly focusing on those capable of processing high-resolution images. As indicated in Table 1, our CLIP variant outperforms the normal resolution open-source models and GPT-4 but still lags behind the high-resolution models and GPT-4o variant. Conversely, our SigLIP variant outperforms all open-source models and GPT-4 variants, achieving competitive performance with GPT-4o, the current top-performing GPT-4 variant. Unlike GPT-4o, which shows a significant disparity between attribute and spatial tasks, our model maintains balanced high scores across both tasks. However, despite our model's competitive performance, it is not as favorable in terms of inference time compared to high-resolution models. Nonetheless, our model offers a significant advantage as it does not require any training, thereby reducing both training costs and resource consumption.

Lastly, we compared our models with SEAL, the current state-of-the-art model on V* Bench. SEAL significantly outperforms our models across all tasks, likely due to its ability to zoom in and slide, mimicking how humans search for small details in large images. However, SEAL's processing time for one sample is lengthy because it passes the prompt multiple times to the LLM. Additionally, SEAL's training procedure involves different modality inputs and requires external models, which makes it less accessible to the general public. In contrast, our models employ only plug-and-play modules and a text encoder as external architecture, achieving moderate inference time while maintaining competitive accuracy.

4.2.2 Multi-V* Bench

Method	Attibute (%)↑	OCR (%)↑	Existence (%)↑	Count (%)↑	Overall (%)↑	Inference Time (sample/s)↓
Open-source models						
Mantis-CLIP ‡[22]	35.0	20.0	35.0	30.0	30.0	**0.749**
Mantis-SigLIP ‡[22]	50.0	50.0	25.0	<u>50.0</u>	43.75	<u>0.774</u>
High resolution models						
LLaVA-NeXT-Interleave [10]	50.0	45.0	45.0	35.0	43.75	0.852
Private models						
GPT-4V Preview [36]	30.0	60.0	45.0	25.0	40.0	-
GPT-4 Turbo [36]	35.0	60.0	45.0	20.0	40.0	-
GPT-4o [36]	**70.0**	<u>65.0</u>	**55.0**	**55.0**	**61.25**	-
GPT-4o Mini [36]	45.0	<u>65.0</u>	<u>50.0</u>	15.0	43.75	-
Ours-CLIP	<u>60.0</u>	50.0	45.0	45.0	50.0	8.099
Ours-SigLIP	45.0	**75.0**	45.0	40.0	<u>51.25</u>	7.194

Table 2: Evaluation of vision language models on Multi-V* Bench. Best results are in bold, and the second-best performances are underlined. ‡ indicates our base model with their respective vision encoder.

Similar to our evaluation on the V* Bench, our models significantly enhance the performance of baseline models by large margins: 20.0% and 7.5% on the overall score for the CLIP and SigLIP variants, respectively. For the CLIP variant, improvements are observed across all tasks, with the most notable gains in the OCR task, likely due to the LLM's enhanced ability to analyze text in greater detail. Similarly, the SigLIP variant shows the greatest improvement in the OCR task. However, slight decreases in performance are noted in the attribute and count tasks. A plausible explanation for the attribute task is that the target objects are dispersed among patches, and with a limited number of selected patches, the LLM may not capture the entire object, complicating attribute identification. In the counting task, the model tends to perform better when viewing the entire image rather than individual patches. Additionally, there is a risk of missing some target objects to be counted since only a limited number of patches per image are processed. Like in the V* Bench tests, processing times are considerably longer due to the model handling multiple images and requiring multiple passes through the LLM.

We also compared our model's performance against other VLMs capable of processing multiple images. As shown in Table 2, our models outperform the open-source model LLaVA-NeXT-Interleave [10] by 6.25% and 7.5% for the CLIP and SigLIP variants, respectively. However, LLaVA-NeXT achieves more favorable inference time since it does not require the LLM to be utilized twice. In comparison with the GPT-4 family, our models outperform all variants except for GPT-4o. GPT-4o exceeds the performance of our models in every task except the OCR task, where our models remain competitive. While GPT-4o achieves the highest overall score, our models rank second and third, demonstrating robust capabilities across a range of tasks.

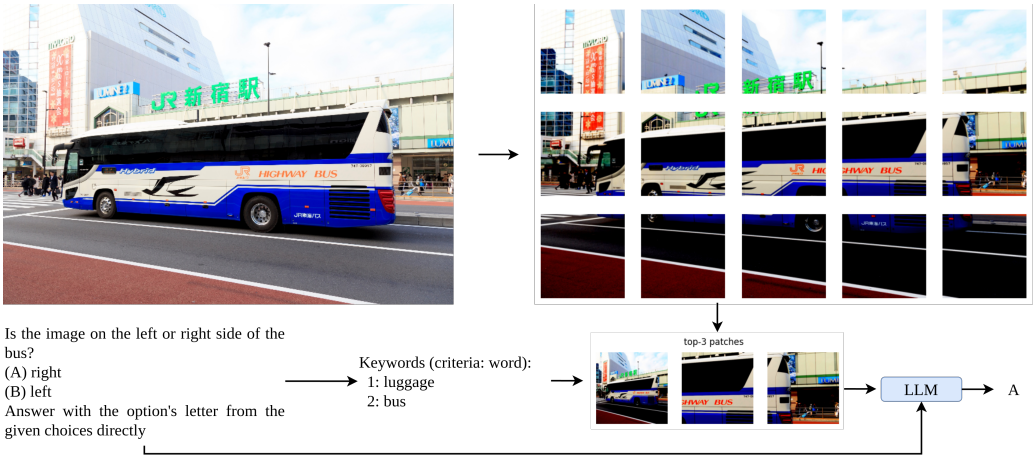
4.3 Qualitative Results

4.3.1 V* Bench

We selected one sample from each task, similar to the images shown in Figure 14. Figure 16 visualizes how our model functions. As described in Section 3.1, we resize and crop the image to produce patches, each matching the vision encoder's native resolution. The keyword extractor then processes the prompt, extracting the keyword "little girl's shirt." Subsequently, the patch selector module selects the top-k patches based on the highest similarity scores; here, we set $k = 2$. As depicted in Figure 16a, one of the top-k patches contains the little girl, demonstrating that the vision encoder effectively identifies the correct patches based on the extracted keywords. These patches are then fed into the LLM to obtain the answer to the question, which is "pink".



(a) Sample from the attribute task.



(b) Sample from the spatial task.

Figure 16: Qualitative study of our models on V* Bench [49].

In the case of the spatial task, which involves determining the relative positions of two objects, two keywords are extracted. If there are multiple keywords, denoted as n_k, the number of extracted patches will be $\left\lceil \frac{k}{n_k} \right\rceil$ per keyword. For instance, if we set $k = 3$ in Figure 16b, then 2 patches for "bus" and another 2 patches for "luggage" will be selected and passed to the LLM.

4.3.2 Multi-V* Bench

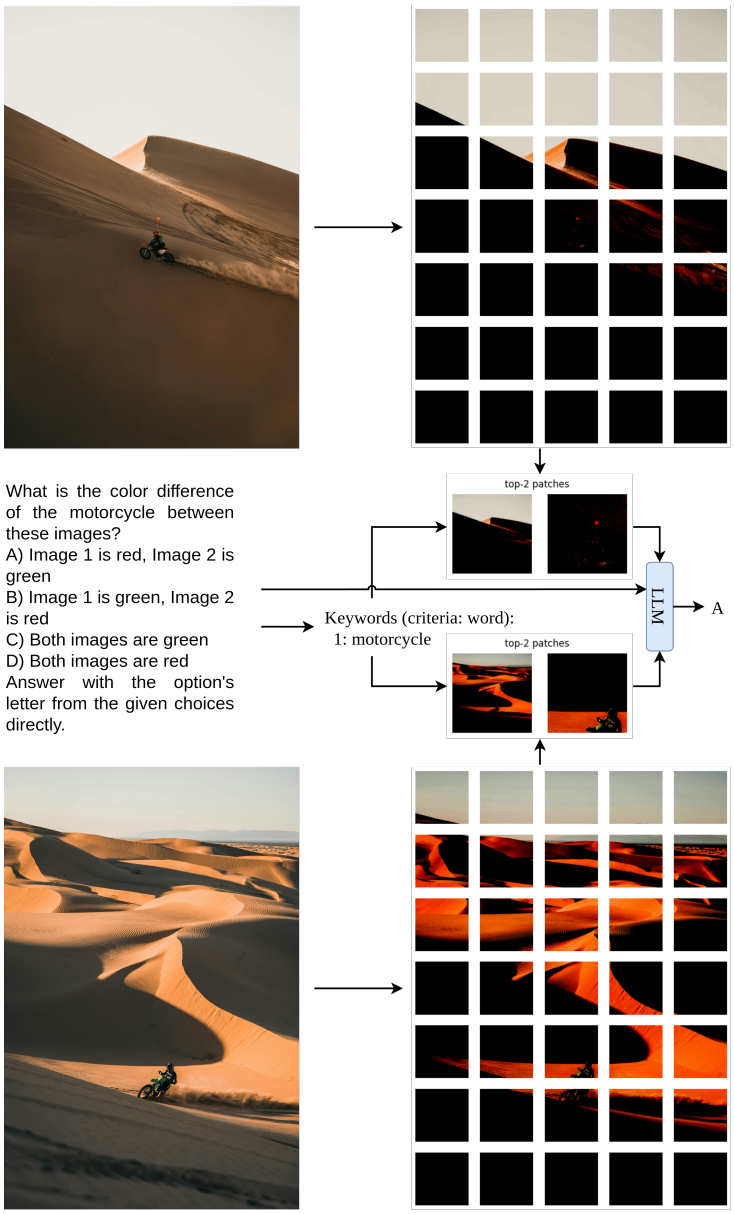


Figure 17: Qualitative study of our models on Multi-V* Bench [49].

We selected a sample similar to the one shown in Figure 15. Figure 17 illustrates how our model operates with multiple images. As with a single image, the images are first cropped into patches, and then the top-k patches are selected. Here, we set $k = 1$, and as illustrated, the patch selector module chooses the patch containing the object "motorcycle." It's important to note that the top-k patches selected are k from each image, resulting in a total of $(k+1) \times \text{num\_image}$ patches.

4.4 Ablation Study - V* Bench

In this section, we will examine how each variable influences the scores and inference time. Throughout this discussion, the top-k variable will always be positioned on the horizontal axis, as this makes it easier to interpret: a higher k should logically lead to an increased score, or at

least reach an asymptote when it includes all the desired patches. Additionally, it's important to note that the scores displayed in the upcoming figures represent the mean across all variables not being specifically observed.

4.4.1 Vision Encoder

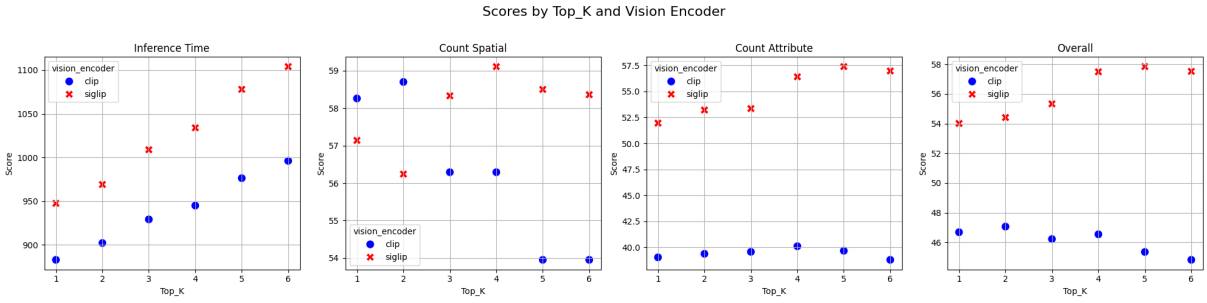


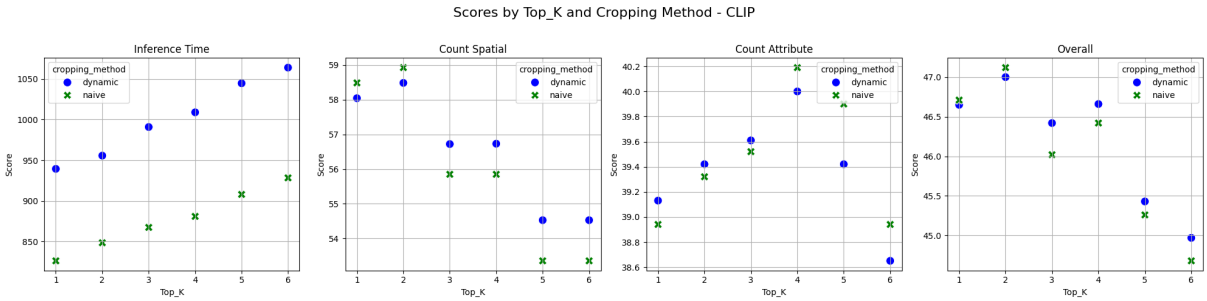
Figure 18: How top-k affects the inference time and scores for different vision encoder.

First of all, we observe that increasing the number of observed patches leads to higher scores across all tasks. This makes sense, as it raises the probability of including the desired patches. However, a higher k also results in longer inference time due to the larger number of image tokens processed. Regarding the performance of the vision encoders, the SigLIP variant significantly outperforms the CLIP variants across all tasks. This finding aligns with several previous studies [23, 57] and is consistent with the trend that the SigLIP vision encoder generally exhibits superior performance compared to the CLIP vision encoder. Nevertheless, the SigLIP variant has longer inference time as it produces more image tokens compared to the CLIP variant (728 vs 576), which likely contributes to its enhanced performance.

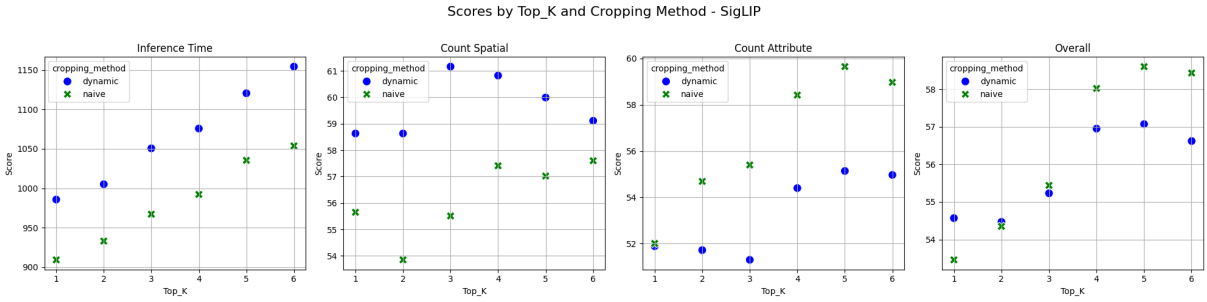
4.4.2 Cropping Method

As shown in Figure 19, for the CLIP variant, the dynamic cropping method incurs longer inference time due to its exhaustive search for the right combinations of n_w and n_h. It tends to perform better in spatial scores, likely because the naive method may split or cut the target object across several patches, while the dynamic method tends to capture the entire target object in a single patch. For attribute scores, the naive method underperforms compared to the dynamic method at lower k values, probably for similar reasons. However, as k increases, the naive method surpasses the dynamic method, possibly because it gathers patches of a scattered object, allowing the LLM to view the object more completely. Another factor could be that the naive method, despite not maintaining the aspect ratio, preserves more detail than the dynamic method.

In the SigLIP variant, the trend is similar to that of the CLIP variant, with the dynamic method requiring more time for inference. However, the naive method consistently scores better in attribute recognition. This might be due to its ability to better crop the target object, likely aided by a higher native resolution. Like in the CLIP variant, the spatial score is higher with the dynamic method, possibly because its larger crop sizes have a better chance of capturing the queried objects in one patch, which simplifies positional interpretation for the LLM.



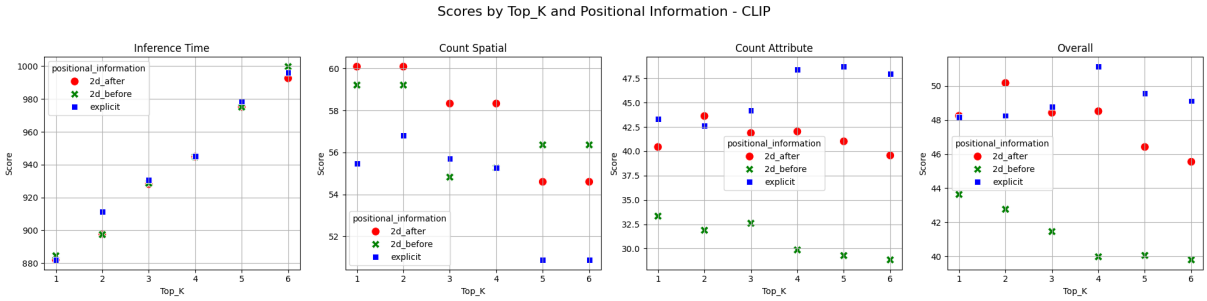
(a) CLIP variant.



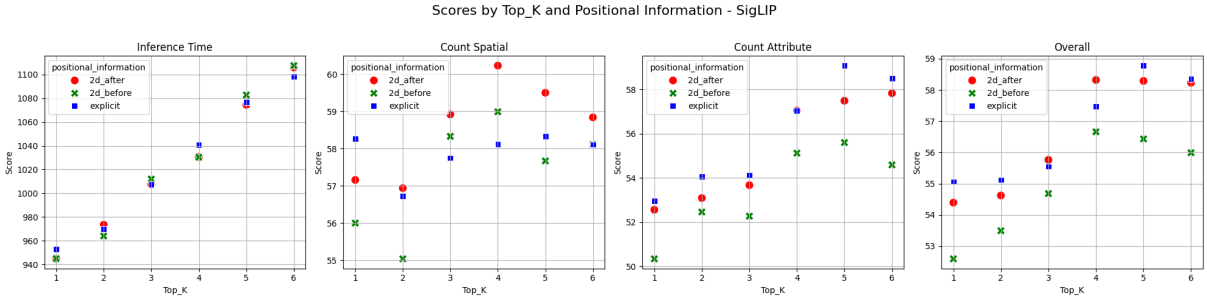
(b) SigLIP variant.

Figure 19: How top-k and cropping method affect the inference time and scores for different vision encoder.

4.4.3 Positional Information



(a) CLIP variant.



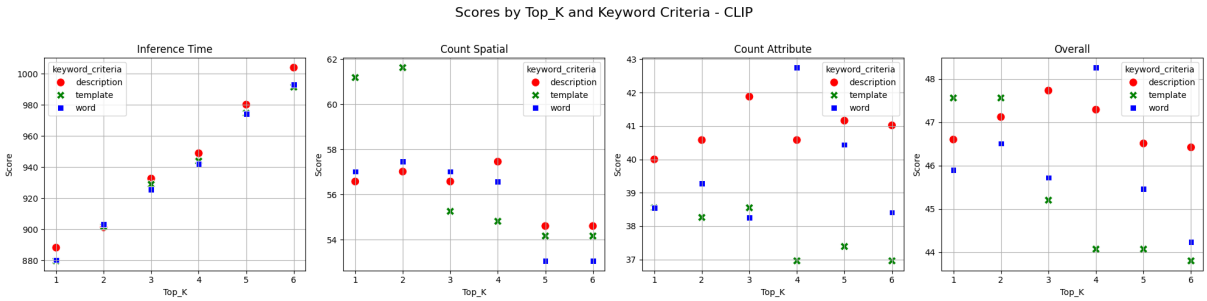
(b) SigLIP variant.

Figure 20: How top-k and positional information affect the inference time and scores for different vision encoder.

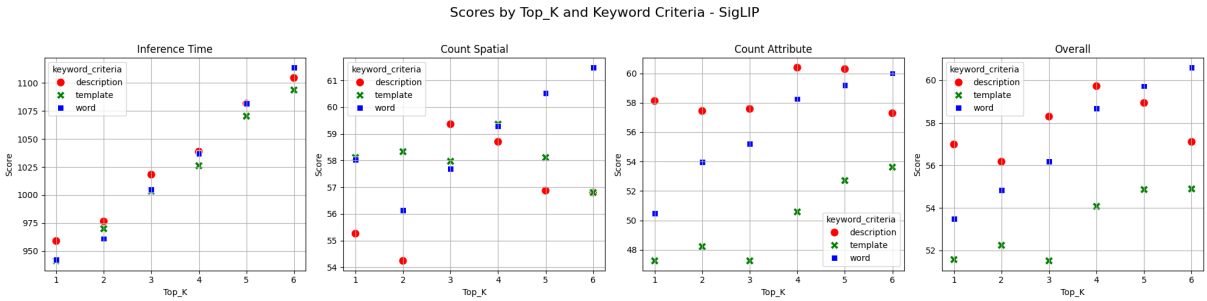
For the CLIP variant, there is no significant difference in inference time between the methods, despite the explicit method generating more tokens. This small increase in tokens does not substantially impact inference time. Spatial scores, which should reflect the modules' ability to correctly identify patches and provide accurate positional information, are highest with the 2d_after method across most k values. However, this method does not surpass the explicit method in attribute scores. This suggests that while 2d_after is effective at providing positional information, it may not accurately identify the correct patches, which is crucial for high scores in both tasks. Therefore, the impact of positional information on spatial scores is unclear. Meanwhile, the explicit method excels in attribute scores, possibly because other forms of positional information are interpreted as noise, given that the LLM was not trained with this data.

Similarly, in the SigLIP variant, inference time are comparable across methods, mirroring the trends observed in the CLIP variant. The 2d_after method dominates in spatial scores but does not outperform the explicit method in attribute scores. This reinforces the conclusion that the relationship between positional information and spatial scores remains indeterminate.

4.4.4 Keyword Criteria



(a) CLIP variant.



(b) SigLIP variant.

Figure 21: How top-k and keyword criteria affect the inference time and scores for different vision encoder.

For the CLIP variant, the description method takes longer for inference because it uses more words for patch selection, but it also yields better results for attribute and overall scores. This aligns with previous research [33] that found descriptions improve results in text and fine-grained image matching. However, similar to the findings with positional information, there seems to be no clear relationship between keyword criteria and spatial score.

In contrast, the SigLIP variant shows different trends from the CLIP variant where the description method does not have the longest inference time. Surprisingly, the word keyword

method, which uses only the keyword for patch selection, takes longer when k is higher. This is counterintuitive since it should be the quickest. This anomaly might be caused by external factors such as the device's condition during inference (e.g., overheating leading to unpredictable performance). Additionally, the word method achieves the highest scores in both attribute and overall categories when k is at its maximum. As far as the author understands, the image-text matching dataset used to pre-train SigLIP primarily utilizes single words rather than templates, which likely explains why the word method outperforms others in this variant.

4.4.5 Best Configuration

Vision Encoder	Top K	Cropping Method	Positional Information	Keyword Criteria	Attribute	Spatial	Overall
CLIP	4	Dynamic	Explicit	Word	52.17	53.95	52.88
SigLIP	6	Naive	Explicit	Word	63.48	61.84	62.83

Table 3: Best configurations for CLIP and SigLIP variants on V* Bench [49].

The table above shows the configurations that yield the highest overall scores for the CLIP and SigLIP variants, respectively. It is evident that the best configurations align with the trends depicted in figure from previous sections, where explicit positional information and the word keyword criteria perform best for both vision encoders. However, there is a notable difference in the efficiency of the cropping methods between the two variants, which could be attributed to the differing native resolutions of the vision encoders. Additionally, SigLIP achieves its best performance when k is at its maximum, whereas CLIP only requires four selected patches. This discrepancy could be due to the redundancy of selected patches; in the case of CLIP, a larger number of patches may confuse the LLM, resulting in poorer performance.

4.5 Ablation Study - Multi-V* Bench

We conducted an ablation study on Multi-V* Bench similar to what was described in Section 4.4. As in the previous study, the top-k variable is consistently placed on the horizontal axis, and the scores displayed in the figures represent the mean of all variables not actively being considered.

4.5.1 Vision Encoder

Overall, it is observed that increasing k leads to a higher overall score, as shown in Figure 22. This makes sense as selecting more patches increases the likelihood of including those that contain the target object. However, a higher k also results in longer inference time due to the processing of additional image tokens. Interestingly, unlike in the V* Bench, the CLIP variant exhibits longer inference time compared to the SigLIP variant. This is unexpected since SigLIP typically generates more image tokens. One plausible explanation is that the image resolutions in this benchmark might align better with SigLIP's native resolution, thus optimizing the dynamic cropping process and reducing time consumption compared to those in the V* Bench.

On the other hand, the SigLIP variant outperforms the CLIP variant in all tasks except for the attribute task, contrasting with results from the V* Bench. Several factors could contribute

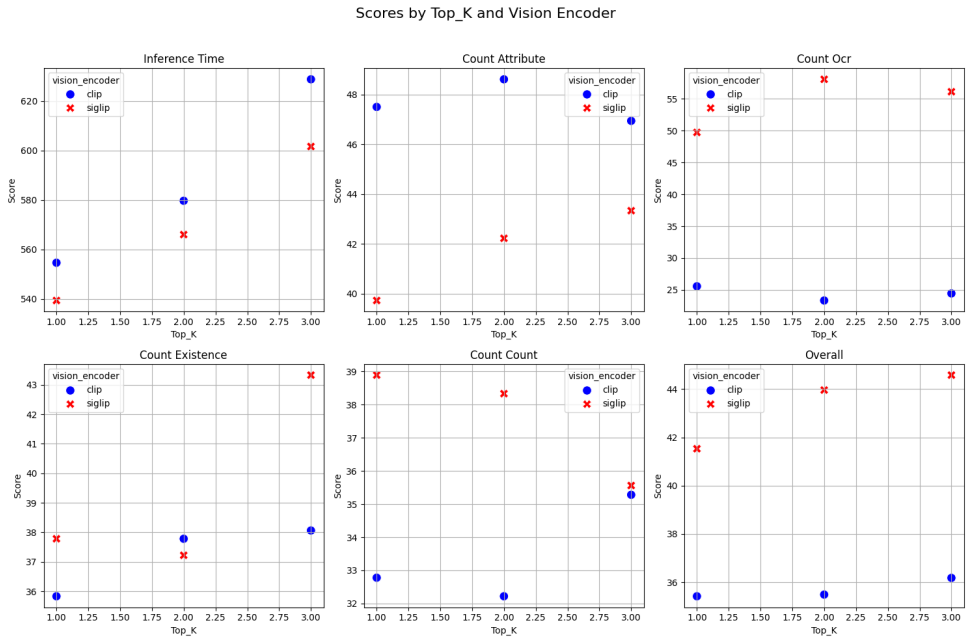


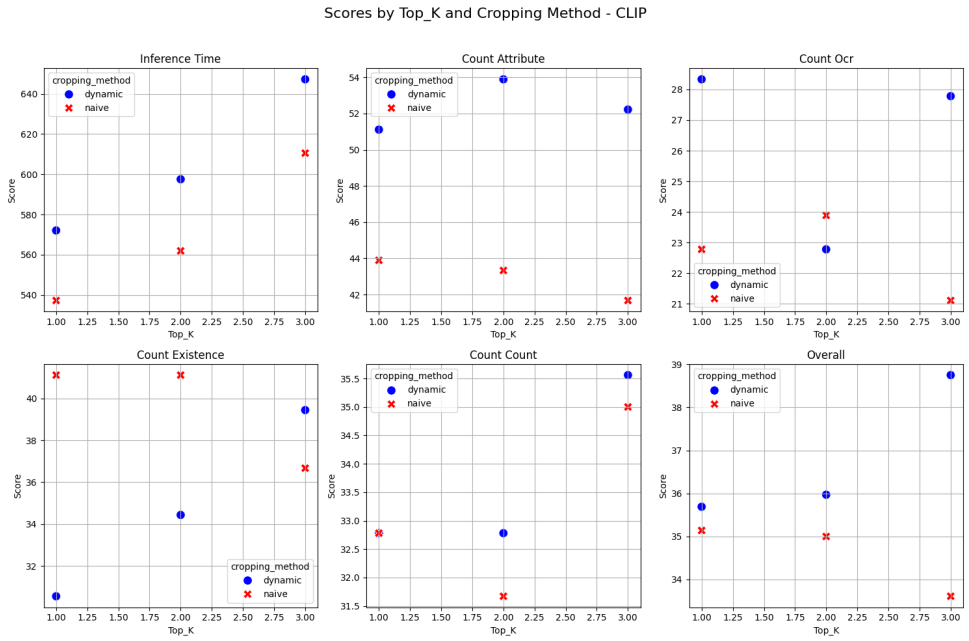
Figure 22: How top-k affects the inference time and scores for different vision encoder.

to this discrepancy, possibly including mismatches between the cropping method, the native resolution of the encoder, and the higher average resolution of the images compared to those in the V* Bench.

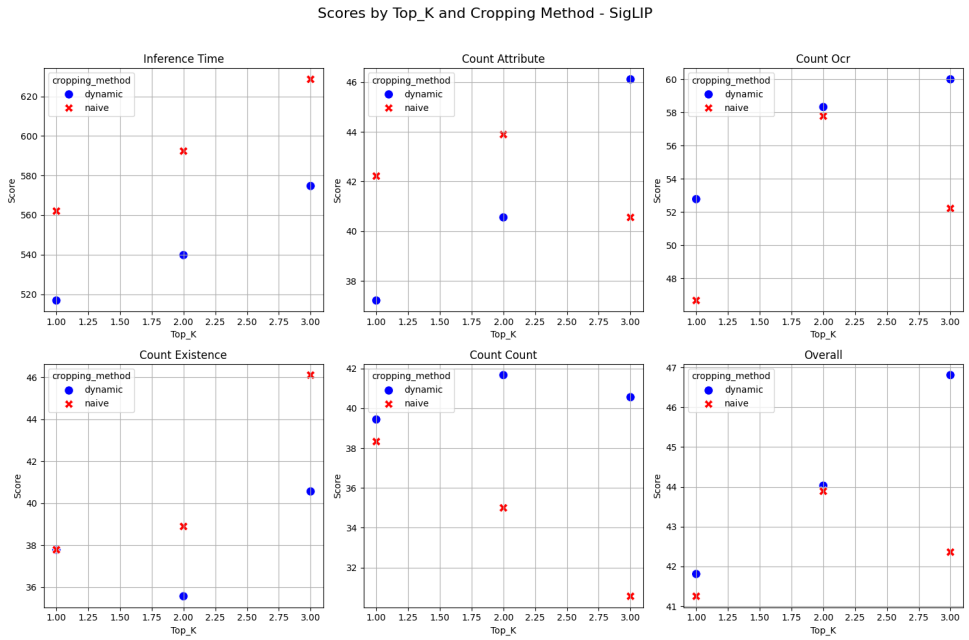
4.5.2 Cropping Method

As shown in Figure 23, for the CLIP variant, the dynamic method requires longer inference times due to its exhaustive search for the right combinations of n_w and n_h. This method consistently outperforms the naive method across almost all tasks, particularly in the attribute and OCR tasks for every k. These trends are consistent with findings from the V* Bench, where the dynamic method not only took longer for inference but also provided better performance than the naive method.

Conversely, the SigLIP variant displays an unusual trend in inference time, with the naive method taking longer. A plausible explanation for this is that the image resolutions more closely match the native resolution, making the process less exhaustive than in the CLIP variant. In terms of performance, the SigLIP variant follows a similar pattern to the CLIP variant, excelling in almost every task except for the existence task. This discrepancy may be due to the dynamic method cropping the image to a larger size, which complicates the patch selector's ability to perform image-text contrast effectively, as larger images tend to include more noise.



(a) CLIP variant.



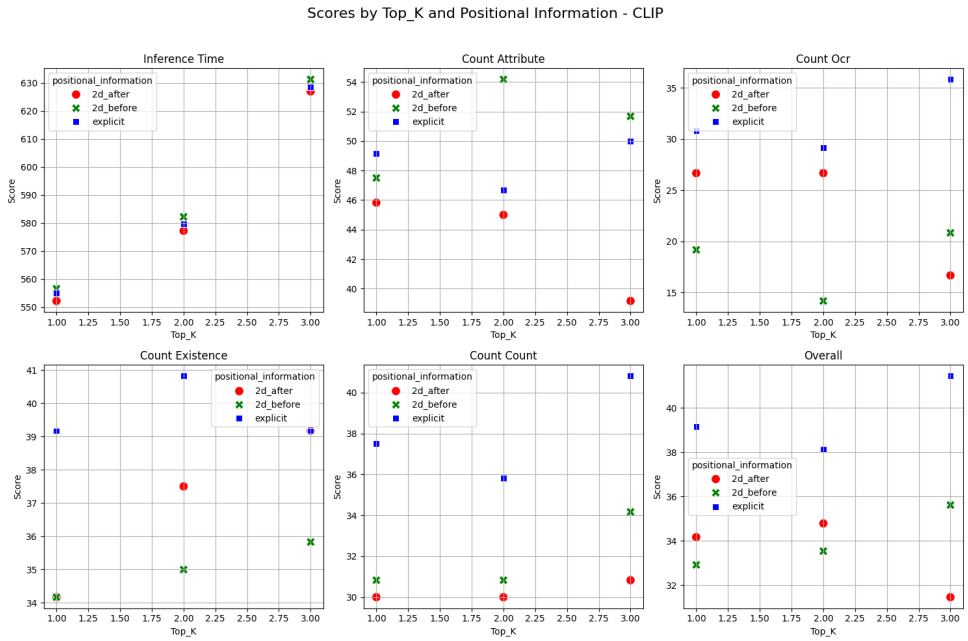
(b) SigLIP variant.

Figure 23: How top-k and keyword criteria affect the inference time and scores for different vision encoder.

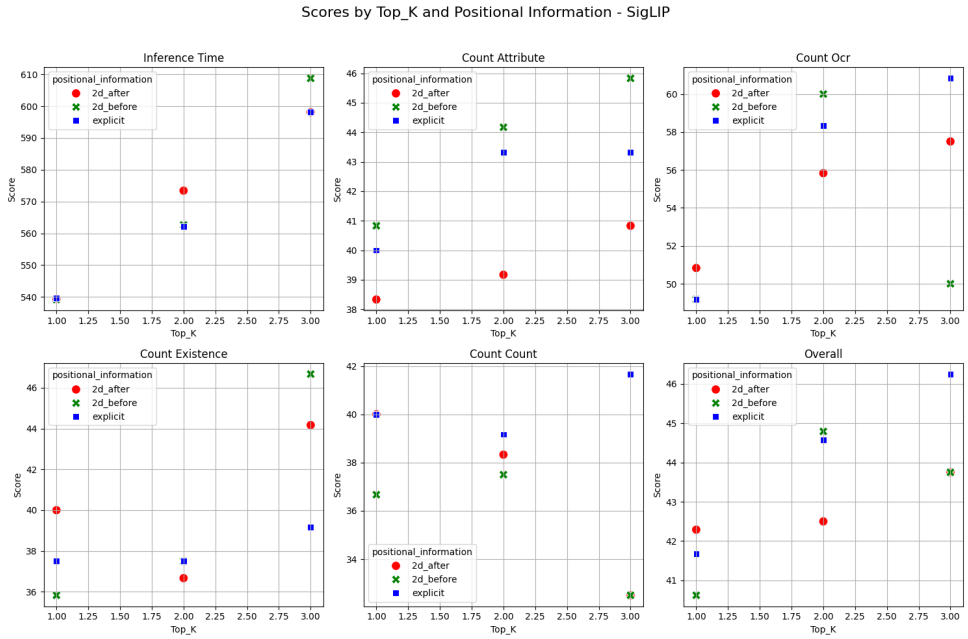
4.5.3 Positional Information

Since this benchmark does not include tasks that require identifying the position of target objects, we would expect positional information variables such as 2d_after and 2d_before to perform worse compared to the explicit method. This is because the LLM was not pre-trained with additional positional information, which could introduce noise into the data processed by the LLM. According to Figure 24, the overall score confirms our hypothesis, with the explicit method outperforming the other methods.

However, there are exceptions, such as the existence task in the SigLIP variant, where both the 2d_before and 2d_after methods surpass the explicit method. In these instances, the explicit information may confuse the LLM, leading it to make incorrect selections. Additionally, it is noted that the 2d_before method has longer inference time for both the CLIP and SigLIP variants, contrary to expectations that the explicit method would take longer due to processing more tokens. This anomaly could be attributed to hardware issues, such as GPUs that have been running continuously, potentially causing longer processing times for subsequent inferences.



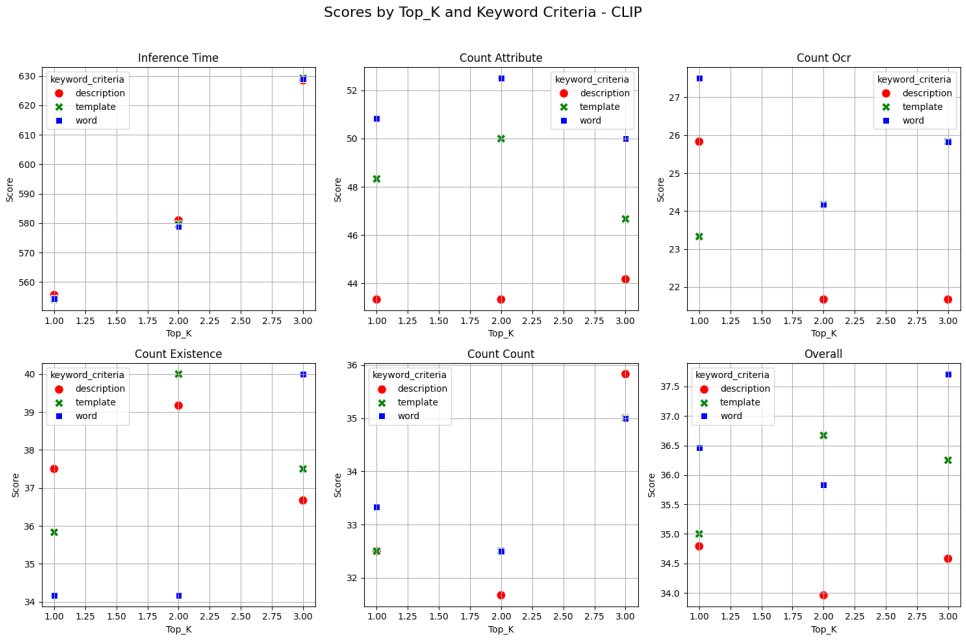
(a) CLIP variant.



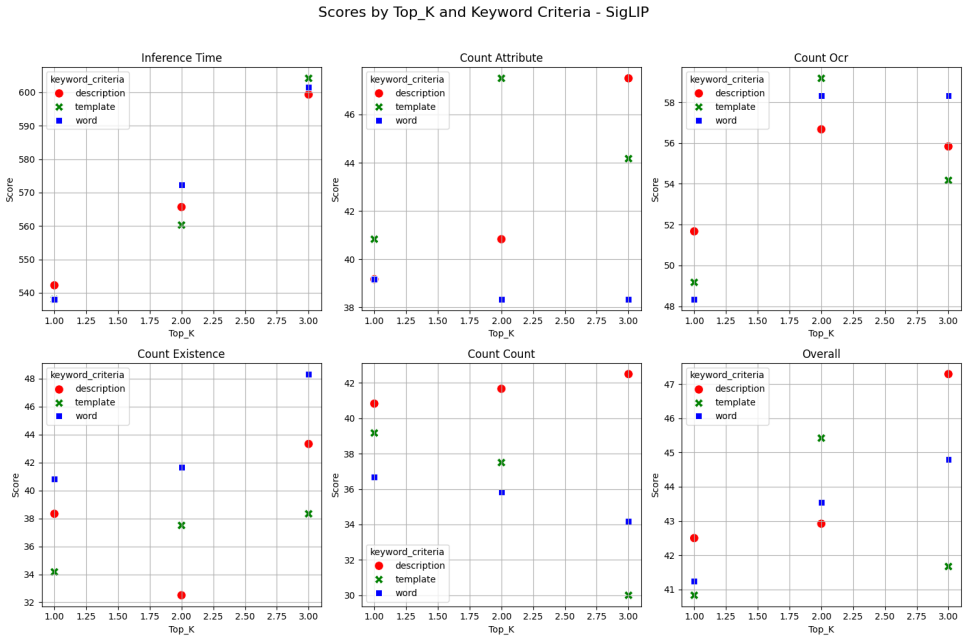
(b) SigLIP variant.

Figure 24: How top-k and keyword criteria affect the inference time and scores for different vision encoder.

4.5.4 Keyword Criteria



(a) CLIP variant.



(b) SigLIP variant.

Figure 25: How top-k and keyword criteria affect the inference time and scores for different vision encoder.

For the CLIP variant, it can be observed that the inference time does not significantly vary between each criterion. In terms of performance, the word variant excels almost in every task, which contrasts with results from V* Bench where the description criterion outperformed the others. One plausible reason could be that the extracted keywords might lack a detailed description. For example, in the OCR task, some questions involve specific text on objects within each image (such as a plate number or brand), which inherently lacks visual features.

This scenario makes it difficult for the patch selector to identify the relevant patches, unlike the word criterion where the extracted keyword may directly correspond to the queried text.

For the SigLIP variant, there is a significant difference in inference time, with the word and template criteria taking longer. This could be reasonable, considering there might be instances where no descriptions are generated, as mentioned earlier. In terms of performance, contrary to the CLIP case, the description criterion outperforms the other criteria in the SigLIP variant. This result contradicts our initial hypothesis that a lack of detailed descriptions would consistently yield the worst performance. Moreover, it also contradicts our argument from the V* Bench, where SigLIP performed best with the word criterion. In my opinion, further exploration and experimentation are necessary to understand the impact of this variable on the model's performance more comprehensively.

4.5.5 Best Configuration

Vision Encoder	Top K	Cropping Method	Positional Information	Keyword Criteria	A	O	E	C	Overall
CLIP	3	Dynamic	Explicit	Word	60.0	50.0	45.0	45.0	50.0
SigLIP	3	Dynamic	Explicit	Word	45.0	75.0	45.0	40.0	51.25

Table 4: Best configurations for CLIP and SigLIP variants on Multi-V* Bench. Here, A, O, E, and C represent the attribute, OCR, existence, and count tasks, respectively.

The table above outlines the best configurations that yield the highest overall scores for both vision encoder variants. The trends align with the ablation figures from previous sections, although Figure 25b indicates that the best keyword criterion for SigLIP is the description criterion. Since the scores in the figure represent averages of all variables not being considered, there may be some deviations from the optimal scores. Additionally, unlike in the V* Bench evaluation, the best cropping method for SigLIP in this analysis is the dynamic method. This difference could be attributed to the varying average resolutions between the two benchmarks. Overall, the configurations are consistent with the findings from the V* Bench.

5 Conclusion

We introduced plug-and-play modules to enhance an existing Vision Language Model's ability to detect small details in high-resolution images. By leveraging the vision encoder's inherent image-text matching capabilities, our modules significantly improved performance on the V* Bench without the need for additional training or external helper models. Additionally, our models outperform both open and closed-source models, achieving the second-best performance on the benchmark. Although our models still fall short of the current state-of-the-art, they reduce inference time by 18 times, striking a balance between accuracy and efficiency.

We also conducted an ablation study to evaluate the impact of various configurations on the performance of our modules. This study included adjustments to the cropping method, criteria for image-text matching, the number of selected patches, the positional information of each patch, and the vision encoder itself. The findings demonstrate that with optimal configurations, our models can secure the second-highest performance on this benchmark.

Since the adopted VLM is capable of processing multiple images, we developed a benchmark similar to V* Bench, but tailored for multiple images and incorporating more diverse tasks, named Multi-V* Bench. Testing our models on this new benchmark showed a similar trend to V* Bench, where our models significantly enhanced the performance of the existing models. Additionally, we compared our models against other multi-image VLMs, such as LLaVA-NeXT and the GPT-4-V family, and found that our models outperformed these, securing the second and third highest performances.

We also conducted an ablation study on Multi-V* Bench, which revealed trends similar to those observed in V* Bench, albeit with some differences in behavior. In conclusion, our modules effectively identify small details in high-resolution images, both in single and multi-image settings.

References

[1] Marah I Abdin et al. "Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone". In: *CoRR* abs/2404.14219 (2024). DOI: 10.48550/ARXIV.2404.14219 (cited on p. 7).

[2] Bo Adler et al. "Nemotron-4 340B Technical Report". In: *CoRR* abs/2406.11704 (2024). DOI: 10.48550/ARXIV.2406.11704 (cited on p. 7).

[3] Alfred V Aho. *Algorithms for finding patterns in strings, Handbook of theoretical computer science (vol. A): algorithms and complexity*. 1991 (cited on p. 14).

[4] Meta AI. "Llama 3". In: (2024) (cited on pp. 1, 3, 7, 8, 18).

[5] Mistral AI. "Mistral". In: (2024) (cited on p. 7).

[6] Joshua Ainslie et al. "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Association for Computational Linguistics, 2023, pp. 4895–4901. DOI: 10.18653/V1/2023.EMNLP-MAIN.298 (cited on p. 7).

[7] Jean-Baptiste Alayrac et al. "Flamingo: a Visual Language Model for Few-Shot Learning". In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by Sanmi Koyejo et al. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/960a172bc7fbf0177ccccbb411a7d800-Abstract-Conference.html (cited on p. 7).

[8] Reza Yazdani Aminabadi et al. "DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale". In: *CoRR* abs/2207.00032 (2022). DOI: 10.48550/ARXIV.2207.00032 (cited on p. 18).

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1409.0473 (cited on p. 7).

[10] Kaichen Zhang Bo Li. *Next: Stronger llms supercharge multimodal capabilities in the wild*. 2024. URL: https://llava-vl.github.io/blog/2024-05-10-llava-next-stronger-llms/ (cited on pp. 21, 22).

[11] Junbum Cha et al. "Honeybee: Locality-enhanced Projector for Multimodal LLM". In: *CoRR* abs/2312.06742 (2023). DOI: 10.48550/ARXIV.2312.06742 (cited on p. 7).

[12] Kezhen Chen et al. "Dragonfly: Multi-Resolution Zoom Supercharges Large Visual-Language Model". In: *CoRR* abs/2406.00977 (2024). DOI: 10.48550/ARXIV.2406.00977 (cited on pp. 1, 9, 21).

[13] Hyung Won Chung et al. "Scaling Instruction-Finetuned Language Models". In: *CoRR* abs/2210.11416 (2022). DOI: 10.48550/ARXIV.2210.11416 (cited on p. 1).

[14] Wenliang Dai et al. "InstructBLIP: Towards General-purpose Vision-Language Models with Instruction Tuning". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh et al. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/9a6a435e75419a836fe47ab6793623e6-Abstract-Conference.html (cited on p. 1).

[15] Tri Dao. "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning". In: *CoRR* abs/2307.08691 (2023). DOI: 10.48550/ARXIV.2307.08691 (cited on p. 18).

[16] Google Deepmind. “Gemini”. In: (2023) (cited on p. 1).

[17] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YicbFdNTTy (cited on p. 4).

[18] Chunjiang Ge et al. “ConvLLaVA: Hierarchical Backbones as Visual Encoder for Large Multimodal Models”. In: *CoRR* abs/2405.15738 (2024). DOI: 10.48550/ARXIV.2405.15738 (cited on pp. 7, 9).

[19] Sylvain Gugger et al. *Accelerate: Training and inference at scale made simple, efficient and adaptable.* https://github.com/huggingface/accelerate. 2022 (cited on p. 20).

[20] Tanmay Gupta and Aniruddha Kembhavi. “Visual Programming: Compositional visual reasoning without training”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 2023, pp. 14953–14962. DOI: 10.1109/CVPR52729.2023.01436 (cited on p. 1).

[21] Dan Hendrycks and Kevin Gimpel. “Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units”. In: *CoRR* abs/1606.08415 (2016). arXiv: 1606.08415 (cited on p. 6).

[22] Dongfu Jiang et al. “MANTIS: Interleaved Multi-Image Instruction Tuning”. In: *CoRR* abs/2405.01483 (2024). DOI: 10.48550/ARXIV.2405.01483 (cited on pp. 1, 18, 21, 22).

[23] Siddharth Karamcheti et al. “Prismatic VLMs: Investigating the Design Space of Visually-Conditioned Language Models”. In: *CoRR* abs/2402.07865 (2024). DOI: 10.48550/ARXIV.2402.07865 (cited on pp. 9, 25).

[24] Alexander Kirillov et al. “Segment Anything”. In: *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 2023, pp. 3992–4003. DOI: 10.1109/ICCV51070.2023.00371 (cited on p. 1).

[25] Hugo Laurençon et al. “What matters when building vision-language models?” In: *CoRR* abs/2405.02246 (2024). DOI: 10.48550/ARXIV.2405.02246 (cited on pp. 1, 21).

[26] Bo Li et al. “MIMIC-IT: Multi-Modal In-Context Instruction Tuning”. In: *CoRR* abs/2306.05425 (2023). DOI: 10.48550/ARXIV.2306.05425 (cited on p. 1).

[27] Jeffrey Li et al. “DataComp-LM: In search of the next generation of training sets for language models”. In: *CoRR* abs/2406.11794 (2024). DOI: 10.48550/ARXIV.2406.11794 (cited on p. 7).

[28] Junnan Li et al. “BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models”. In: *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 19730–19742. URL: https://proceedings.mlr.press/v202/li23q.html (cited on pp. 1, 7).

[29] Yanwei Li et al. “Mini-Gemini: Mining the Potential of Multi-modality Vision Language Models”. In: *CoRR* abs/2403.18814 (2024). DOI: 10.48550/ARXIV.2403.18814 (cited on p. 21).

[30] Haotian Liu et al. “Improved Baselines with Visual Instruction Tuning”. In: *CoRR* abs/2310.03744 (2023). DOI: 10.48550/ARXIV.2310.03744 (cited on pp. 1, 20, 21).

[31] Haotian Liu et al. “Visual Instruction Tuning”. In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh et al. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html (cited on pp. 1, 6).

[32] Gen Luo et al. “Feast Your Eyes: Mixture-of-Resolution Adaptation for Multimodal Large Language Models”. In: *CoRR* abs/2403.03003 (2024). DOI: 10.48550/ARXIV.2403.03003 (cited on p. 21).

[33] Sachit Menon and Carl Vondrick. “Visual Classification via Description from Large Language Models”. In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/forum?id=jlAjNL8z5cs (cited on p. 27).

[34] Thomas Mesnard et al. “Gemma: Open Models Based on Gemini Research and Technology”. In: *CoRR* abs/2403.08295 (2024). DOI: 10.48550/ARXIV.2403.08295 (cited on p. 7).

[35] Matthias Minderer et al. “Simple Open-Vocabulary Object Detection”. In: *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part X*. Ed. by Shai Avidan et al. Vol. 13670. Lecture Notes in Computer Science. Springer, 2022, pp. 728–755. DOI: 10.1007/978-3-031-20080-9_42 (cited on p. 1).

[36] OpenAI. “GPT-4 Technical Report”. In: *CoRR* abs/2303.08774 (2023). DOI: 10.48550/ARXIV.2303.08774 (cited on pp. 1, 20–22).

[37] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 8024–8035. URL: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html (cited on p. 20).

[38] *Pexels*. 2013. URL: https://www.pexels.com/ (cited on p. 19).

[39] Reiner Pope et al. “Efficiently Scaling Transformer Inference”. In: *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*. Ed. by Dawn Song, Michael Carbin, and Tianqi Chen. mlsys.org, 2023. URL: https://proceedings.mlsys.org/paper%5C_files/paper/2023/hash/c4be71ab8d24cdfb45e3d06dbfca2780-Abstract-mlsys2023.html (cited on p. 7).

[40] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8748–8763. URL: http://proceedings.mlr.press/v139/radford21a.html (cited on pp. 1, 3, 4, 18).

[41] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. URL: http://jmlr.org/papers/v21/20-074.html (cited on p. 1).

[42] Noam Shazeer. “GLU Variants Improve Transformer”. In: *CoRR* abs/2002.05202 (2020). arXiv: 2002.05202 (cited on p. 7).

[43] Jianlin Su et al. “RoFormer: Enhanced Transformer with Rotary Position Embedding”. In: *CoRR* abs/2104.09864 (2021). arXiv: 2104.09864 (cited on p. 7).

[44] *Unsplash*. 2013. URL: https://unsplash.com/ (cited on p. 19).

[45] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (cited on pp. 7, 16).

[46] Thomas Wolf et al. “HuggingFace’s Transformers: State-of-the-art Natural Language Processing”. In: *CoRR* abs/1910.03771 (2019). arXiv: 1910.03771 (cited on p. 20).

[47] Chenfei Wu et al. “Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models”. In: *CoRR* abs/2303.04671 (2023). DOI: 10.48550/ARXIV.2303.04671 (cited on p. 1).

[48] Chenfei Wu et al. “Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models”. In: *CoRR* abs/2303.04671 (2023). DOI: 10.48550/ARXIV.2303.04671 (cited on p. 1).

[49] Penghao Wu and Saining Xie. “V*: Guided Visual Search as a Core Mechanism in Multimodal LLMs”. In: *CoRR* abs/2312.14135 (2023). DOI: 10.48550/ARXIV.2312.14135 (cited on pp. 1, 18, 21, 23, 24, 28).

[50] An Yang et al. *Qwen2 Technical Report*. 2024. arXiv: 2407.10671 [cs.CL] (cited on p. 7).

[51] Linli Yao et al. “DeCo: Decoupling Token Compression from Semantic Abstraction in Multimodal Large Language Models”. In: *CoRR* abs/2405.20985 (2024). DOI: 10.48550/ARXIV.2405.20985 (cited on p. 7).

[52] Andy B. Yoo, Morris A. Jette, and Mark Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-39727-4 (cited on p. 20).

[53] Xiaohua Zhai et al. “Sigmoid Loss for Language Image Pre-Training”. In: *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 2023, pp. 11941–11952. DOI: 10.1109/ICCV51070.2023.01100 (cited on pp. 3, 6, 18).

[54] Biao Zhang and Rico Sennrich. “Root Mean Square Layer Normalization”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 12360–12371. URL: https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html (cited on p. 7).

[55] Susan Zhang et al. “OPT: Open Pre-trained Transformer Language Models”. In: *CoRR* abs/2205.01068 (2022). DOI: 10.48550/ARXIV.2205.01068 (cited on p. 1).

[56] Lianmin Zheng et al. “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena”. In: *CoRR* abs/2306.05685 (2023). DOI: 10.48550/ARXIV.2306.05685 (cited on pp. 1, 7).

[57] Baichuan Zhou et al. “TinyLLaVA: A Framework of Small-scale Large Multimodal Models”. In: *CoRR* abs/2402.14289 (2024). DOI: 10.48550/ARXIV.2402.14289 (cited on pp. 9, 25).

[58] Deyao Zhu et al. “MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models”. In: *CoRR* abs/2304.10592 (2023). DOI: 10.48550/ARXIV.2304.10592 (cited on p. 1).

IMAC	Guidance from Keyword for Identifying Small Details in High-Resolution Images with Vision-Language Models	September 2024	Farros Alferro