The supplementary materials are arranged as follows: In Sec.A, we illustrate additional method details of the LLaFS framework. In Sec.B, we provide more implementation details of our method. In Sec.C, we present more experimental results to further demonstrate the effectiveness of our method and designs. In Sec.D, we showcase some examples of the key steps from input to output of the LLaFS framework.

A. More Method Details of LLaFS

A.1. Complete Input for ChatGPT

In our proposed expert-guide framework for instruction refinement, we input ChatGPT with a text prompt describing the task requirements to achieve ambiguity detection and discriminative attributes generation (see Fig.3 of main paper for examples). In practice, the complete ChatGPT input is formed by combining this text prompt with a ***format control prompt*** that explicitly specifies the format of the output we expect from ChatGPT. When the output format is unified and fixed, it is easier for us to extract the ambiguous classes $\{[\text{a-class}]_i\}_{i=1}^{N_{ac}}$ and discriminative attributes $\{[\text{d-att}]_i\}_{i=1}^{N_d}$ from ChatGPT's feedback automatically and efficiently. Specifically, the complete ChatGPT inputs for ambiguity detection and discriminative attributes generation are written as:

Ambiguity Detection: Except for [class], which classes also have [partial-attributes]? Please answer in the format of: the following classes also have them: A, B, C, ..., , where A, B and C are the name of classes. If there is no such a class, reply 'no'.

Discriminative Attributes Generation: What does [class] look different from [a-classes]? Please answer in the format of: [class] has A, B, C,..., where A,B and C are noun phrases to describe the difference of [class] compared to [a-classes].

Discriminative Attributes Generation (from the second iteration onwards): Apart from [all-discriminative-attributes], tell me more differences in appearance between [class] and [a-classes]. Please answer in the format of: [class] has A, B, C,..., where A,B and C are noun phrases to describe more differences of [class] compared to [a-classes] apart from the given ones.

A.2. More Details of Pseudo-sample-based Curriculum Pretraining

A.2.1 Contour Generation Method

When generating pseudo images, we first randomly generate a contour within an image region, and the area surrounded by this contour is considered as the foreground within the target class. This contour is generated as a Bézier curve constructed by 10 randomly-generated control points. We will release code to illustrate it in detail.

A.2.2 Details of Pseudo Support-query Generation

The detailed method for generating a support-query pair can be summarized as the following steps:

Step 1: Support foreground-background partition. We first randomly generate a contour within an image region. The area surrounded by this contour is considered as the foreground within the target class, while the regions outside the contour are treated as the background. For the background, we use random contours to divide it into multiple subregions to simulate the diverse backgrounds encountered in real images. The number of subregions is randomly selected from 1 to 5.

Step 2: Support noise filling. We randomly generate an array $m_{sf} \in \mathbb{R}^3$ within the value range [0, 255], and utilize it as the RGB mean to generate a Gaussian noise for filling the support foreground region. Subsequently, for each subregion of the support background, we randomly generate another array $m_{sb} \in \mathbb{R}^3$ as the mean to generate a Gaussian noise for filling this subregion. We constrain the random generation space of m_{sb} to satisfy the distance constraint $||m_{sb} - m_{sf}|| \in [a, b]$, where a, b are two adjustable parameters. By adjusting the values of a and b, we can manage the difference between the foreground and background within each synthetic image. Sec.A.2.3 illustrates how to adjust them in different pretraining steps.

Step3: Query foreground-background partition by adjusting from support. To ensure that the support foreground and query foreground have similar shapes so that they can reflect the same category, the contour used to generate the query foreground is adjusted based on that used for generating the support foreground. Specifically, we first add a standard Gaussian noise to the ten control points that are used to generate the support foreground contour. Subsequently, a noised contour is generated from these noised control points, followed by the random rotation and scaling between [0.5, 1.5] for further adjustment. After that, we randomly place the resulted contour in another position of the image, and the region enclosed by which is regarded as the query foreground. Finally, we use the same approach as the support background to partition the query background region.

Step4: Query noise filling by adjusting from support. Using the same method as for the support generation, we randomly generate arrays $m_{qf} \in \mathbb{R}^3$ and $m_{qb} \in \mathbb{R}^3$ and use them as RGB means to generate Gaussian noises, which are then applied to fill the query foreground and each subregion of the query background. To ensure that the support foreground and query foreground have similar internal features so that they can reflect the same category, we constrain the random generation space of m_{qf} to satisfy the distance constraint $||m_{qf} - m_{sf}|| \in [c, d]$, where c and d are adjustable

parameters. For the background's m_{qb}, we impose two constraints to determine its random generation space: (1) similar to the support background, we constrain the difference between the query background and query foreground by $||m_{qb} - m_{qf}|| \in [a, b]$. (2) To ensure that query foreground is the most similar region to the support foreground in the query image, we further constrain the difference between the query background and the support foreground to be greater than the difference between the query foreground and the support foreground. This is achieved by constraining $||m_{qb} - m_{sf}|| > ||m_{qf} - m_{sf}||$. Under these constraints, we randomly generate m_{qf} and m_{qb} to serve as the means of noises, which are used to fill different regions to obtain the pseudo query image.

A.2.3 Details of Curriculum Pretraining

During pretraining, we incrementally raise the task's difficulty from the following two aspects:

(1) Image understanding. During pretraining, by controlling the difference between mean values of different filled noise, we gradually increase the difference in foreground between support and query, while reducing the internal difference between foreground and background within each image. This makes it more challenging for LLM to perform few-shot guidance and partition foreground-background areas as pretraining progresses. We implement this strategy by adjusting the parameters a, b, c, d introduced in the previous section.

Specifically, the interval $[a, b]$ constrains the difference between the foreground and background within an image. Therefore, during the pretraining process, to reduce this difference, we gradually decrease the values of a, b until a eventually reaches 0. Denoting the total number of pretraining steps as N_p ($N_p = 60K$ in our experiments), the values of a_n and b_n at step n are formulated as:

$$\begin{aligned} a_n &= a_0 - \frac{n.a_0}{N_p}, \\ b_n &= a_n + b_0 - a_0, \end{aligned} \tag{1}$$

where a_0 and b_0 are the hyper-parameters that define the initial values of a and b in the first step of pretraining.

The interval $[c, d]$ constrains the difference between the support foreground and query foreground. Therefore, during the pretraining process, to enlarge this difference, we gradually increase the values of c and d, making c to be increased from 0 to c_{N_p} as the step progresses from 0 to N_p. In this way, the values of c_n and d_n at step n are formulated as:

$$\begin{aligned} c_n &= \frac{n.c_{N_p}}{N_p}, \\ d_n &= c_n + d_{N_p} - c_{N_p}, \end{aligned} \tag{2}$$

where c_{N_p} and d_{N_p} are the hyper-parameters that define the final values of a and b in the last step of pretraining.

In this approach, a_0, b_0, c_{N_p} and d_{N_p} are predefined hyper-parameters, which are respectively set to 100, 150, 50, and 100 in our framework. Note that, our experiments demonstrate that ***the performance of LLaFS is NOT sensitive to these hyper-parameters***. See Sec.C and Table.2 for details.

(2) Polygon generation. During the pretraining stage, we randomly provide the coordinates of M points in the instruction and let the LLM to predict the coordinates of the remaining $16-M$ points. M is decreased by 1 every $N_p/30$ steps in the first half pretraining process with $N_p/2$ steps. By doing so, we gradually decrease the value of M from 15 to 0. This means that the model receives fewer hints and is required to predict more vertex coordinates as pretraining progresses. Consequently, the pretraining difficulty gradually increases, ultimately reaching the task of predicting all 16 points for segmentation. In the last half pretraining process, we keep $M = 0$ for pretraining.

A.2.4 Instruction in Pretraining.

In the pretraining stage, the instruction for inputting into LLM is written as: For the target object in a query image that has the same class as the support image foreground, output coordinates of a 16-point polygon that encloses the object. These points should be arranged in a clockwise direction and the format of their coordinates is ((x1,y1),(x2,y2),...,(x16,y16)). The coordinate value should be within [image size]. For support image [pseudo support image], the foreground is [support foreground]. For the target object in the query image [pseudo query image], the output should be [masked-gt]. What is the remaining points? Here, [masked-gt] refers to retaining part of the ground truth vertices for the 16-point polygon as hints, while replacing the remaining parts to be predicted with a [mask] token as in [2].

A.3. Refinement Network

With the instruction as input, the LLM can predict the coordinates of a 16-point polygon. We use 1 to fill the area enclosed by the polygon and 0 to fill the area outside the polygon. In this way, we obtain a binary segmentation mask denoted as **M**. To rectify the imprecision caused by the polygon representation of object edges, we further introduce a refinement network to obtain a more refined segmentation result. As shown in Fig.1, this refinement network follows a similar structure to Mask2Former [1], comprising a pixel decoder that progressively increases the sizes of query image feature maps and a masked transformer decoder for optimizing the queries. **M** is used as the mask for the masked attention in the transformer decoder. Readers can refer to Sec3.21 of [1] for more details of masked attention.

Note that, compared to the vanilla Mask2Former, our method does not employ a heavy transformer to construct

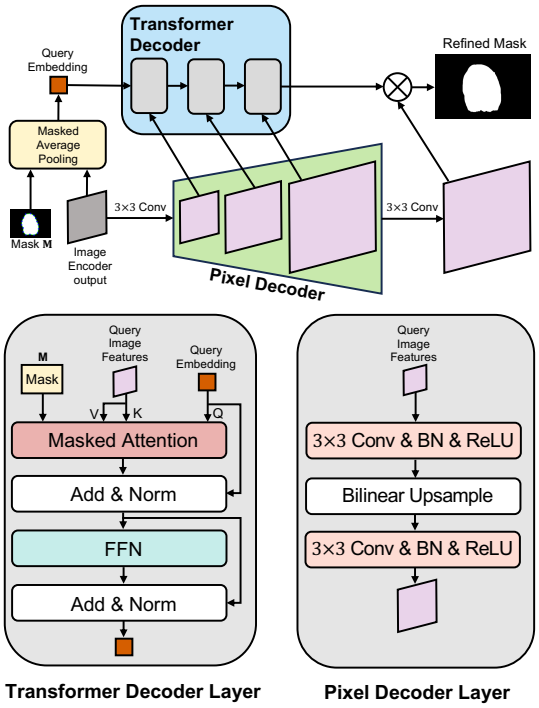


Figure 1. Structure of the refinement network. This network is lightweight, comprising only 6 convolution layers and 3 attention layers.

the pixel decoder; instead, we use a simple structure composed of a small number of convolution and bilinear upsampling layers. Moreover, we do not iteratively apply the transformer decoder but use it only once. These modifications reduce the computational complexity, making our refinement network to be very lightweight with only 6 convolution layers and 3 attention layers. As the output from the LLM already achieves pretty good results, using this lightweight network for refinement is completely sufficient.

A.4. Expanding LLM Vocabulary with Coordinate Tokens

We expand the vocabulary of LLM by adding 384 coordinate tokens denoted as ([c-0], [c-1], ..., [c-383]), where [c-i] represents the coordinate value i. This design makes our model more efficient, requiring fewer tokens for the input and output of LLM.

A.5. Loss Function

Using the Vertex Coordinates outputted by the LLM to construct a binary mask as the input for the refinement network is a non-differentiable process. Therefore, we employ two separate loss functions to train the refinement network and the remaining components of LLaFS, respectively. In this way, the overall loss can be written as:

$$\mathcal{L} = \mathcal{L}_{llm} + \mathcal{L}_{ref}, \tag{3}$$

where $\mathcal{L}_{llm}$ denotes the loss for training fully-connected layers and LLM, $\mathcal{L}_{ref}$ denotes the loss for training the refinement network. For $\mathcal{L}_{llm}$, we first use bipartite matching to align the LLM-predicted outputs with each object in the ground truth, then we use cross-entropy loss to compute $\mathcal{L}_{llm}$. For $\mathcal{L}_{ref}$, we use cross-entropy loss with online hard examples mining (OHEM) strategy to compute it.

A.6. Extension to Multi-shot Setting

In the main paper, we introduce LLaFS under the one-shot setting. The method for the multi-shot setting can be easily extended from the one-shot method. Specifically, for each support image, we extract a set of visual tokens and a region-attribute corresponding table using the method illustrated in main paper. These pieces of information from all K support images are then incorporated into the following instruction for feeding into the LLM: For each object within the class [class] in an image, output coordinates of a 16-point polygon that encloses the object. These points should be arranged in a clockwise direction. The output should be a tuple in the format of (c1, c2, ..., cn), where cn is the coordinates for the n-th object and its format should be ((x1,y1),(x2,y2),...,(x16,y16)). The coordinate value should be within [image size]. To accomplish this task, you can refer to the following properties of [class]: [class] has [attributes]. For example, for image [support image 1], the output should be [support ground truth 1], because in these regions, $[\text{coord}]_{s_1}$ is $[\text{cor}]_1$, $[\text{coord}]_{s_2}$ is $[\text{cor}]_2$..., $[\text{coord}]_{s_{N_r}}$ is $[\text{cor}]_{N_r}$; ...; for image [support image K], the output should be [support ground truth K], because in these regions, $[\text{coord}]_{s_1}$ is $[\text{cor}]_1$, $[\text{coord}]_{s_2}$ is $[\text{cor}]_2$..., $[\text{coord}]_{s_{N_r}}$ is $[\text{cor}]_{N_r}$. For image [query image], what is the output?

B. More Implementation Details

Image Encoder. ResNet50 from the CLIP is used as the image encoder. In ResNet50, the output features from stage 3 and stage 4 are resized to 1/8 of the input size and concatenated with the output features from stage 2. This combined feature is used as the input for the Q-former and the pixel decoder in the refinement network.

Q-former. The Q-former has 8 layers with the dimension of 384. The input for the text transformer in the Q-former is 'a photo of [class].'.

Refinement Network. Feature dimension in the refinement network is 128.

Generation of Region-attribute Corresponding Table. Generating region-attribute corresponding table requires additional time due to the use of ChatGPT. To prevent this additional computation from affecting training efficiency, we pre-generate the table for each image before training and include it as part of the dataset that can be directly used

Method	PASCAL-5^i	COCO-20^i
Painter [4]	64.5	32.8
SegGPT [5]	83.2	56.1
LLaFS	88.3	64.2

(a) Comparison with SegGPT and Painter.

Method	mIoU
LLaFS	74.2
LLaFS w/o LLM	62.3

(b) Effectiveness of large language models.

LLM	mIoU
Llama2	69.8
Code Llama	74.2

(c) Llama VS Code Llama as the LLM of LLaFS.

Method	mIoU
LLaFS	74.2
LLaFS w/ curriculum polygon generation in training	74.6

(d) Curriculum polygon generation in the training stage.

Method	mIoU
Our Curriculum Strategy	74.2
Masking with a Fixed Ratio $\lambda = 0.25$	69.5
Masking with a Fixed Ratio $\lambda = 0.5$	71.5
Masking with a Fixed Ratio $\lambda = 0.75$	71.1

(e) Curriculum strategy vs fixed-ratio masking for polygon generation.

Method	mIoU
LLaFS	74.2
LLaFS w/o curriculum strategy in image understanding	72.0
LLaFS w/o curriculum strategy in polygon generation	68.1
LLaFS w/o increasing SF-QF difference	72.9
LLaFS w/o reducing F-B difference	72.6

(f) Ablation results of curriculum pretraining. SF, QF, F, B refer to support foreground, query foreground, foreground, background.

Table 1. **More experimental results.**

in all experiments. We will release these tables to facilitate future research.

Data Augmentation. We employ random horizontal flipping, random noise padding, random cropping, and random resizing for data augmentation. Note that to prevent the random cropping from causing the mismatch between the region-attribute corresponding table and the augmented support image, we constrain the range of random cropping when augmenting each support image to ensure that the foreground region within the target class is not cropped.

Other Training Settings. The batch size is 32. The input image size is (384, 384).

C. More Experimental Results

Comparison with SegGPT and Painter. In addition to LLaFS, SegGPT [5] and Painter [4] can also achieve few-shot segmentation through in-context learning. Fig.1a presents the comparison results with these methods. For a fair comparison, we follow SegGPT by combining different segmentation datasets for training and allow the categories in training to cover the categories in testing. It is observed that on both PASCAL-5^i and COCO-20^i, our approach can achieve significant advantages. These results demonstrate that our LLaFS can perform in-context-based few-shot segmentation more effectively, which is benefited from the rich prior knowledge contained in LLM and our carefully-designed fine-grained multi-modal demonstration examples.

Effectiveness of Large Language Models Thanks to the rich prior knowledge and powerful few-shot capabilities, large language models (LLM) play a crucial role in ensuring the high effectiveness of our LLaFS framework. To demonstrate this, we remove the LLM from LLaFS and validate the effectiveness of a few-shot segmentation model that is composed of the remaining parts of LLaFS.

Specifically, to ensure that the few-shot segmentation can be performed by only using the remaining components, we make the following modifications: (1) When extracting visual tokens from the support image using the Q-former, the vanilla cross-attention that interacts the learned queries with support image features is replaced with the masked attention as in [1]. This change ensures that the obtained support tokens are only related to the foreground region where the target category is located. (2) After the Q-former, we add a cross-attention to interact support tokens with query tokens. This allows query tokens to perceive reference information from the support foreground. The query tokens obtained through this step are used as the input query embeddings for the transformer decoder in the refinement network, which produces the final segmentation result.

As shown in Table.1b, although the other network components remain largely unchanged, removing LLM significantly decreases mIoU by 11.9%. This demonstrates the crucial role of the LLM in our LLaFS framework.

Llama VS Code Llama. In LLaFS, we employ Code Llama instead of the vanilla Llama as the large language model. As shown in Table.1c, the performance of using Code Llama is 4.4% better than using Llama. This improvement could be attributed to the fact that Code Llama has been fine-tuned on the code generation dataset, so it is more skilled in generating structured data with fixed formats, such as the segmentation results in our task.

Ablation of Curriculum Pretraining. In the curriculum pretraining strategy, we gradually increase the difficulty of the pretraining tasks from two aspects: (1) image understanding and (2) polygon generation. As shown in Table.1f, Not applying the curriculum strategy in each of these two aspects decreases the mIoU by 2.2% and

$(a_0, b_0, c_{N_p}, d_{N_p})$	mIoU
(100, 150, 50, 100)	74.2
(75, 125, 75, 125)	74.0
(125, 175, 25, 75)	74.3
(100, 150, 75, 125)	74.0
(75, 125, 50, 100)	74.0

Table 2. Different settings for hyper-parameters $(a_0, b_0, c_{N_p}, d_{N_p})$ of the pseudo-sample-based curriculum pretraining mechanism.

6.1%, respectively. To increase the difficulty of image understanding, we employ two methods when synthesizing support-query pairs: (1) increasing the difference between support foreground and query foreground, and (2) reducing the difference between foreground and background within each image. Removing each of these two methods decreases the mIoU by 1.3% and 1.6%, respectively. These results demonstrate the effectiveness of our designs in the approach.

Curriculum Strategy VS Fixed-Ratio Masking for Polygon Generation. We further test a masked strategy used in [3–5], in which we randomly provide a fixed ratio λ of vertex coordinates in the instruction, and let the model predict the remaining vertices. We test three values for λ: 0.25, 0.5, and 0.75. As shown in Table.1e, the performances of all these methods are worse than our curriculum strategy. These results demonstrate the effectiveness of our approach, showing the importance of dynamically increasing the learning difficulty during the pretraining process.

Curriculum Polygon Generation in the Training Stage. In addition to employing the curriculum polygon generation on synthetic images during the pretraining stage, we also test the further usage of this strategy to realistic data during the training stage. As shown in Table.1d, we observe that such a modification cannot significantly improve performance. One possible reason could be that the model has acquired sufficient ability to generate 16-point coordinates through pretraining on the pseudo samples, so it no longer requires the continued use of this strategy in the subsequent training stage.

Hyper-parameter Settings for Pseudo-sample-based Curriculum Pretraining. As discussed in detail in Sec.A.2.3, our proposed pseudo-sample-based curriculum pretraining involves four hyper-parameters $(a_0, b_0, c_{N_p}, d_{N_p})$. The results for different combinations of these hyper-parameters are presented in Table.2. It can be observed that our method can consistently achieve excellent and similar results across different $(a_0, b_0, c_{N_p}, d_{N_p})$ settings. These results demonstrate that the performance of LLaFS is NOT sensitive to these hyper-parameters.

Ablation for Threshold α. We use a hyper-parameter α as the threshold to construct the region-attribute corresponding table in LLaFS (See Sec.3.2.2 and Eq.1 of main paper for details). Fig.2 presents the results when using different values as α. It can be observed that both excessively small and large values of α can decrease the mIoU. This could be because an overly small α may result in the incorrect region-attribute match, while an excessively large α may lead to missed matches, both of which can adversely affect the quality of the generated table. When $0.20 \leq \alpha \leq 0.22$, the model can consistently achieve stable and high performance.

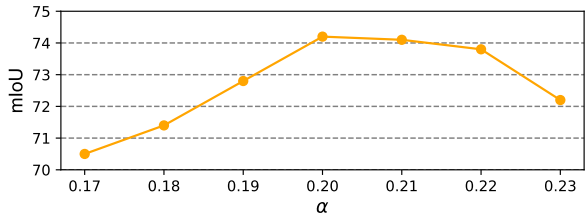


Figure 2. Using different values of threshold α in Eq.1 of main paper.

D. Examples of Key Steps from Input to Output

We provide two examples to show the key steps from input to output in LLaFS, including network input, class attributes, region-attribute corresponding table, instruction refinement, complete instruction, LLM output, and final result. These examples are presented in the following 5 pages.

References

[1] Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1290–1299, 2022. 2, 4

[2] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. 2

[3] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022. 5

[4] Xinlong Wang, Wen Wang, Yue Cao, Chunhua Shen, and Tiejun Huang. Images speak in images: A generalist painter for in-context visual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6830–6839, 2023. 4

[5] Xinlong Wang, Xiaosong Zhang, Yue Cao, Wen Wang, Chunhua Shen, and Tiejun Huang. Seggpt: Towards segmenting everything in context. In *Proceedings of the IEEE/CVF Inter-*

national Conference on Computer Vision, pages 1130–1140, 2023. 4, 5

Example 1

(1) Input Image


Support Image


Support Ground Truth


Query Image

(2) Class Attributes

What does a horse look like? Please answer in the format of: A horse has A, B, C,...,where A,B and C are noun phrases to describe a horse.

A horse has a sleek coat, a long tail, a muscular body, four legs with hooves, a muzzle, expressive eyes.

Class Attributes: $[att]_1$: a sleek coat $[att]_2$: a long tail $[att]_3$: a muscular body $[att]_4$: four legs with hooves $[att]_5$: a muzzle $[att]_6$: expressive eyes

(3) Region-attribute Corresponding Table

 : $[att]_1$ a sleek coat

 : $[att]_6$ expressed eyes

(4) Instruction Refinement

(4.1) Ambiguity Detection

Except for horse, which classes also have (a sleek coat, expressive eyes)? Please answer in the format of: the following classes also have them: A, B, C, ..., , where A, B and C are the name of classes. If there is no such a class, reply 'no'.

The following classes also have them: Cheetah, Panther, Jaguar, Leopard, Tiger, Lion, Cougar, Domestic Cat.

Ambiguous Classes: $[a\text{-}class]_1$: Cheetah $[a\text{-}class]_2$: Panther $[a\text{-}class]_3$: Jaguar $[a\text{-}class]_4$: Leopard $[a\text{-}class]_5$: Tiger $[a\text{-}class]_6$: Lion $[a\text{-}class]_7$: Cougar $[a\text{-}class]_8$: Domestic Cat

(4.2) Discriminative Attributes Generation

What does horse look different from (Cheetah, Panther, Jaguar, Leopard, Tiger, Lion, Cougar, Domestic Cat)? Please answer in the format of: A horse has A, B, C,...,where A,B and C are noun phrases to describe the difference of a horse compared to (Cheetah, Panther, Jaguar, Leopard, Tiger, Lion, Cougar, Domestic Cat).

A horse has a long and flowing mane, a sturdy hoofed structure, pointed ear.

Discriminative Attributes: $[d\text{-}att]_1$: a long and flowing mane $[d\text{-}att]_2$: a sturdy hoofed structure $[d\text{-}att]_3$: pointed ear

(4.3) Refined Table

 : $[att]_1$ a sleek coat

 : $[att]_6$ expressed eyes

 : $[d\text{-}att]_1$ a long and flowing mane

 : $[d\text{-}att]_3$ pointed ear

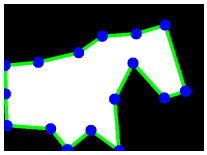
(5) Complete Instruction

For each object within the class horse in an image, output coordinates of a 16-point polygon that encloses the object. These points should be arranged in a clockwise direction. The output should be a tuple in the format of (c1, c2, ..., cn), where cn is the coordinates for the n-th object and its format should be ((x1,y1),(x2,y2),...,(x16,y16)). The coordinate value should be within (0, 384). To accomplish this task, you can refer to the following properties of horse: horse has a sleek coat, a long tail, a muscular body, four legs with hooves, a muzzle, expressive eyes, a long and flowing mane, a sturdy hoofed structure, pointed ear. For example, for image [support image], the output should be (((334, 192), (289, 231), (244, 237), (248, 297), (210, 300), (254, 99), (294, 109), (364, 126), (54, 199), (104, 156), (142, 129), (173, 116), (204, 103), (170, 290), (118, 280), (58, 254))), because in these regions, ((330, 168), (345, 217), (315, 227), (285, 231), (255, 99), (281, 103), (327, 116), (353, 122), (158, 171), (149, 126), (182, 109), (244, 237), (244, 237), (235, 237), (191, 221), (147, 211)) is a sleek coat, ((159, 262), (152, 277), (151, 270), (139, 277), (148, 277), (148, 289), (159, 262), (159, 252), (126, 270), (122, 252), (129, 222), (129, 248), (134, 274), (134, 284), (126, 270), (126, 258)) is expressed eyes , ((119, 235), (119, 235), (119, 235), (119, 235), (117, 280), (133, 139), (160, 174), (160, 174), (48, 221), (66, 175), (66, 175), (83, 162), (83, 162), (74, 270), (57, 265), (48, 221)) is a long and flowing mane, ((89, 254), (116, 254), (106, 254), (96, 254), (89, 254), (94, 215), (94, 215), (94, 215), (48, 221), (48, 221), (56, 215), (88, 254), (88, 254), (87, 254), (74, 251), (74, 251)) is pointed ears. For image [query image], what is the output?

(6) LLM Output and Segmentation Result

(((196, 92), (258, 61), (348, 56), (424, 39), (477, 166), (421, 179), (339, 112), (291, 182), (302, 281), (228, 241), (166, 278), (122, 238), (7, 232), (3, 172), (2, 117), (89, 111)))

LLM Output



Segmentation Mask

Final Result
After Refinement

Example 2

(1) Input Image

Support Image

Support Ground Truth

Query Image

(2) Class Attributes

What does a cow look like? Please answer in the format of: A cow has A, B, C,...,where A,B and C are noun phrases to describe a cow

A cow has a large and sturdy body, a four-chambered stomach, a broad face, curved horns, a tail with a tuft of hair at the end, a snout.

Class Attributes

$[att]_1$: a large and sturdy body $[att]_2$: a four-chambered stomach $[att]_3$: a broad face
$[att]_4$: curved horns $[att]_5$: a tail with a tuft of hair at the end $[att]_6$: a snout

(3) Region-attribute Corresponding Table

 : $[att]_3$ a broad face

 : $[att]_4$ curved horns

 : $[att]_4$ curved horns

 : $[att]_4$ curved horns

 : $[att]_6$ a snout

(4) Instruction Refinement

(4.1) Ambiguity Detection

Except for cow, which classes also have (a broad face, curved horns, a snout) ? Please answer in the format of: the following classes also have them: A, B, C, ..., , where A, B and C are the name of classes. If there is no such a class, reply 'no'.

The following classes also have them: Cheetah, Panther, Jaguar, Leopard, Tiger, Lion, Cougar, Domestic Cat.

Ambiguous Classes

$[a\text{-}class]_1$: Cheetah $[a\text{-}class]_2$: Panther $[a\text{-}class]_3$: Jaguar $[a\text{-}class]_4$: Leopard
$[a\text{-}class]_5$: Tiger $[a\text{-}class]_6$: Lion $[a\text{-}class]_7$: Cougar $[a\text{-}class]_8$: Domestic Cat

(4.2) Discriminative Attributes Generation

What does cow look different from (Wildebeest, Bighorn Sheep, Goat)? Please answer in the format of: A cow has A, B, C,...,where A,B and C are noun phrases to describe the difference of a cow compared to (Wildebeest, Bighorn Sheep, Goat).

A cow has pronounced udders, humped back, domesticated demeanor.

Discriminative Attributes

$[d\text{-}att]_1$: pronounced udders $[d\text{-}att]_2$: humped back $[d\text{-}att]_3$: domesticated demeanor

(4.3) Refined Table

 : $[att]_3$ a broad face

 : $[att]_4$ curved horns

 : $[att]_4$ curved horns

 : $[att]_4$ curved horns

 : $[att]_6$ a snout

Because the newly-acquired discriminative attributes still couldn't find matching regions in the support image, the resulting table after refinement remains to be ambiguous. Therefore, the refinement process is iteratively performed until the ambiguity is eradicated.

(4.4) Discriminative Attributes Generation (2nd Iteration)

Apart from (domesticated demeanor, a loose fold of skin under the neck, humped back), tell me more differences in appearance between cow and (Wildebeest, Bighorn Sheep, Goat). Please answer in the format of: Cow has A, B, C,..., where A,B and C are noun phrases to describe more differences of cow compared to (Wildebeest, Bighorn Sheep, Goat) apart from the given ones.

A cow has a flat face with a pronounced muzzle, a loose fold of skin, streamlined body.

Discriminative Attributes

$[d\text{-}att]_4$: a flat face with a pronounced muzzle $[d\text{-}att]_5$: a loose fold of skin
$[d\text{-}att]_6$: streamlined body

(4.5) Refined Table (2nd Iteration)

 : $[att]_3$ a broad face

 : $[att]_4$ curved horns

 : $[att]_4$ curved horns

 : $[att]_4$ curved horns

 : $[att]_6$ a snout

 : $[d\text{-}att]_4$ a flat face with a pronounced muzzle

 : $[d\text{-}att]_5$ a loose fold of skin

 : $[d\text{-}att]_5$ a loose fold of skin

The framework successfully discovers discriminative attributes [d-att] that can be aligned with the support image. Therefore, the ambiguity is eradicated, the iteration is terminated.

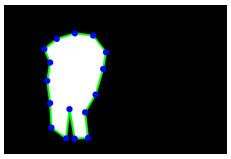
(5) Complete Instruction

For each object within the class cow in an image, output coordinates of a 16-point polygon that encloses the object. These points should be arranged in a clockwise direction. The output should be a tuple in the format of (c1, c2, ..., cn), where cn is the coordinates for the n-th object and its format should be ((x1,y1),(x2,y2),…,(x16,y16)). The coordinate value should be within (0, 384). To accomplish this task, you can refer to the following properties of horse: cow has a large and sturdy body, a four-chambered stomach, a broad face, curved horns, a tail with a tuft of hair at the end, a snout, pronounced udders, humped back, domesticated demeanor, a flat face with a pronounced muzzle, a loose fold of skin, streamlined body. For example, for image [support image], the output should be (((189, 250), (195, 269), (194, 332), (236, 132), (195, 128), (90, 83), (41, 149), (60, 230), (383, 169), (334, 133), (277, 138), (237, 383), (300, 383), (383, 376), (383, 290), (383, 230))), because in these regions, ((285, 165), (256, 208), (187, 207), (164, 207), (142, 89), (158, 123), (176, 125), (217, 131), (41, 162), (52, 128), (65, 87), (101, 69), (134, 79), (109, 240), (61, 242), (46, 199)) is a broad face, a flat face with a pronounced muzzle, ((89, 110), (98, 121), (93, 126), (78, 131), (74, 134), (82, 103), (82, 103), (82, 103), (60, 115), (60, 100), (57, 94), (63, 86), (73, 134), (65, 133), (56, 124), (60, 115)) is curved horns, ((71, 201), (63, 207), (60, 210), (59, 211), (54, 214), (61, 184), (68, 185), (80, 194), (46, 199), (46, 199), (47, 197), (47, 187), (53, 184), (49, 217), (33, 222), (18, 219)) is curved horns, ((90, 150), (94, 166), (83, 173), (82, 195), (65, 87), (82, 103), (93, 126), (84, 146), (41, 158), (42, 146), (46, 138), (52, 128), (63, 86), (46, 199), (42, 169), (41, 162)) is curved horns, ((280, 172), (273, 189), (262, 198), (248, 202), (244, 151), (254, 151), (271, 149), (288, 161), (206, 178), (212, 157), (222, 154), (228, 153), (243, 151), (243, 151), (217, 202), (207, 190)) is a snout, ((383, 218), (383, 252), (349, 266), (321, 266), (302, 158), (326, 160), (363, 156), (383, 184), (168, 215), (263, 204), (273, 196), (284, 172), (301, 158), (281, 261), (259, 261), (215, 256)) is a loose fold of skin, ((383, 184), (383, 206), (350, 205), (340, 205), (331, 207), (336, 161), (356, 157), (383, 162), (282, 183), (283, 164), (305, 158), (321, 159), (327, 207), (317, 208), (303, 210), (263, 204)) is a loose fold of skin. For image [query image], what is the output?

(6) LLM Output and Segmentation Result

((135, 57), (182, 48), (229, 52), (263, 81), (255, 109), (236, 153), (208, 184), (216, 228), (181, 229), (168, 178), (160, 229), (122, 210), (118, 168), (111, 130), (118, 98), (103, 75)))

LLM Output



Segmentation Mask

Final Result
After Refinement