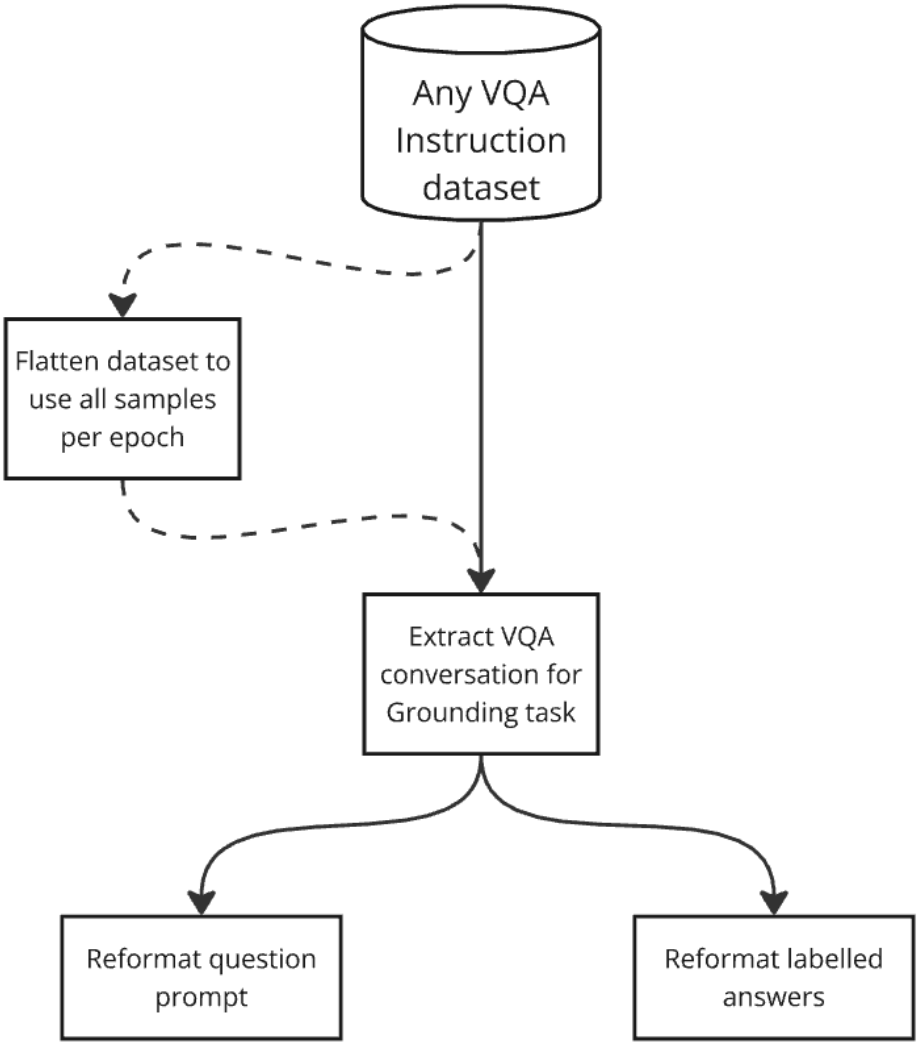
VQA prompt formatting to finetune GeoChat model on Grounding task

Abstract: For every downstream task, prompt-format consistency between any VQA Instruct dataset and the VQA Instruct dataset that a target model was pretrained on is important. As far as I know, there's yet a script or even a package that can do this task for any VQA dataset to any target model with 0% error rate: minigptv has a prompt reformat script which required me to heavily edit it, Llama-Factory package is too much focus on LLM tasks instead of VLM (some script for bounding box rescaling is missing). Given this observation, a fundamental question is which should be our approach for data collection:

1. 1 coding pipeline for all datasets: Resulting in something like Llama-Factory so it will be very hard to debug and still missing desired features to be developed like resizing bounding boxes. The only advantage is that anyone who doesn't have more features can enjoy automation (a really small benefit).
2. 1 technical documentation for each person develops their own coding script: No automation, quick to debug, quick to add on features.

Overall workflow:

Given any VQA Instruction dataset, there are 3 steps with 1 optional step to flatten the dataset to have x1.5-2 training samples per epoch.

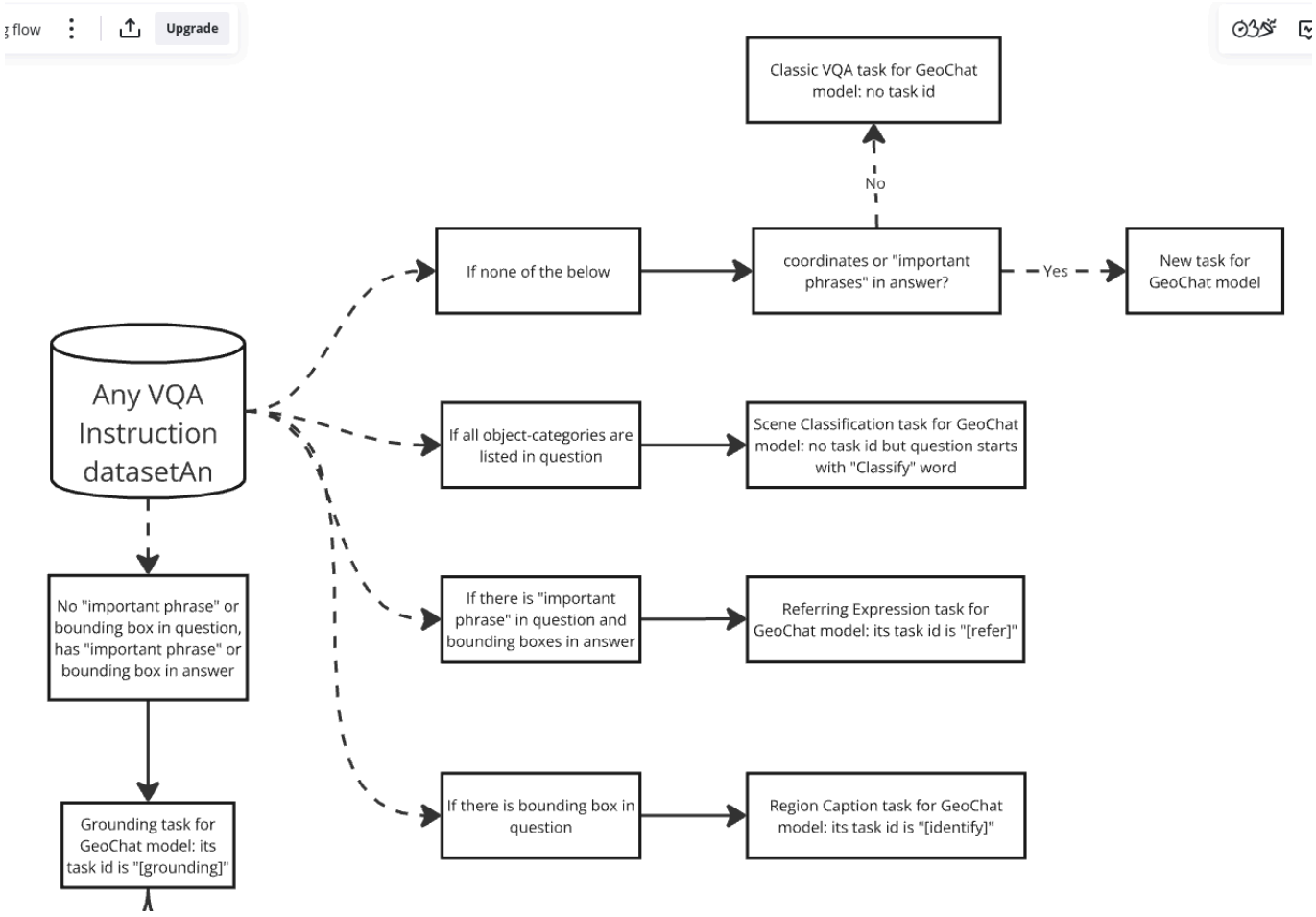


Step 1: Task extraction:

Let's take an example in a ground-captioning task then expand toward other instruction tuning tasks. As far as I know, ground truth QA-pairs for ground-captioning always has:

- Question prompt to describe the entire image in detail. Depending on the VQA dataset, question prompts are sometimes also task-id.
- Answer prompt has some coordinate-format: either horizontal bounding box, quadrilateral box or polygon within a natural text description.

Here are guideline for other instruction tuning tasks.



Step 2: Reformat QA-pair for Florence2 model:

Case example 1: Supposed we are given the original QA pair as below. The steps to do are:

- Rename task-id [grounding] -> <DENSE_REGION_CAPTION> to make sure there is "\n" at the beginning.
- Florence2 bounding box coordinate is normalised in [0,1000] range; original bounding box coordinate is normalised in [0,100] => formatted coordinate = original coordinate / original normalisation range * Florence normalisation range. Also, changing the string pattern "<loc_xx>" -> "<xx>" **can simply be done with GPT**.

```
Original question: '\n[grounding] describe this image as detailed as possible'
Original answer : ' In the image, there are <p>some buildings</p>
{<69><20><77><28>|<90>} with <p>two olive small cars</p>
{<55><15><55><19>|<87>}<delim>{<46><36><46><40>|<88>}  and a <p>white large
small car</p> {<32><28><34><32>|<3>} parked near them. Some trees are visible at
the top of the scene. Nearby, there are <p>some buildings</p>
```

```
{<14><42><24><50>|<19>}<delim>{<17><52><25><60>|<90>}  with an <p>olive large
small car</p> {<14><54><14><58>|<88>} and <p>a white large van</p>
{<22><61><26><63>|<74>} parked together, surrounded by trees. At the left side
of the image, there are <p>some buildings</p> {<27><86><35><92>|<71>}, and some
trees can be seen in the background.'

Formatted question: '<DENSE_REGION_CAPTION>'
Formatted answer: 'some
buildings<loc_690><loc_200><loc_770><loc_280><angle_90>two olive small
cars<loc_550><loc_150><loc_550><loc_190><angle_87><loc_460><loc_360><loc_460><lo
c_400><angle_88>white large small
car<loc_320><loc_280><loc_340><loc_320><angle_3>some
buildings<loc_140><loc_420><loc_240><loc_500><angle_19><loc_170><loc_520><loc_25
0><loc_600><angle_90>olive large small
car<loc_140><loc_540><loc_140><loc_580><angle_88>a white large
van<loc_220><loc_610><loc_260><loc_630><angle_74>some
buildings<loc_270><loc_860><loc_350><loc_920><angle_71>'
```

[Coding output from .ipynb]

Case example 2: Supposed we are given the original QA pair as below, the plans are:

- Change task-id
- Extract important phrase, and for important phrase rescale bounding box [0,1] -> [0, 1000] range, then change string pattern from “xx” into “<loc_xx>” **can simply be done with GPT.**

```
Original question: 'Detect all objects shown in the remote sensing image and
describe using horizontal bounding box'
Original answer : 'There are <p>eight
People</p>[0.36,0.36,0.37,0.40;0.58,0.27,0.59,0.30;0.47,0.49,0.49,0.52;0.45,0.50
,0.47,0.53;0.36,0.67,0.37,0.70;0.35,0.72,0.36,0.74;0.33,0.74,0.34,0.77;0.88,0.15
,0.90,0.17] in the image.'

Formatted question: '<DENSE_REGION_CAPTION>'
Formatted answer: 'eight
People<loc_360><loc_360><loc_370><loc_400><angle_90<loc_580><loc_270><loc_590><l
oc_300><angle_90<loc_470><loc_490><loc_490><loc_520><angle_90<loc_450><loc_500><
loc_470><loc_530><angle_90<loc_360><loc_670><loc_370><loc_700><angle_90<loc_350>
<loc_720><loc_360><loc_740><angle_90<loc_330><loc_740><loc_340><loc_770><angle_9
0<loc_880><loc_150><loc_900><loc_170><angle_90'
```

[Coding result in tokenizer_formatter.ipynb]

From 2 case examples, the general trend I find:

a. For every downstream task,:
    1. Always 1 “important phrase” tokens per sentence
    2. Always “important phrase” sits before coordinates.
    3. Sometimes “important phrases” sit between and tokens like GeoChat, EarthGPT, sometimes not like Florence2.
    4. At baseline, “important phrase” has labelled object-category. Sometimes it also counts number of instances for that labelled object-category, some other times it also contains relative spatial position between object-category-A and object-category-B

b. To make chatGPT good at generating reformatting VQA Python code within 15 mins and prevent sensitive info from being leaked into chatGPT. I do the followings and **keep in mind to never debug code with GPT**:
    1. Identify Dictionary keys per VQA sample, it's mostly in formatted as Dict{'image': relative/path/to/image, 'conversations': [list of many QA-pairs]}. **Some VQA might already be flattened** where it either uses Tuple(image, question, answer) or Dict{'image': relative/path/to/image, 'question': prompt, 'answer': prompt}. This can be done using find_text_in_json.py to view different sections of VQA Instruction dataset formatted as a .json file.
    2. Copy + Paste 1 VQA sample onto text editor and create a desired VQA sample format for GeoChat. **Delete any of our sensitive info**.
    3. From 1 VQA sample, ask chatGPT to split each sentence using (a1) intuition.
    4. Per sentence, ask chatGPT to extract 1 "important phrase" and all bounding boxes of that "important phrase" as 1 string chunk.
    5. Per all-bounding-boxes-per-phrase string chunk, ask chatGPT to extract each bounding box as a list of floating numbers
    6. Per bounding box, rescale every coordinate to GeoChat normalising in [0, 100] range and round to nearest integer. Then cast final integer coordinates as string and concatenate them into a formatted answer.
    7. If an arbitrary VQA dataset uses horizontal bounding box with angle like GeoChat - Great news. If not, there are the following scenarios to take in mind: (**Also take in mind that if another VLM uses another coordinate format other than horizontal bounding box with angle, you have to adjust this step**):
        a. If an arbitrary VQA dataset only has a horizontal bounding box but no angle => assume that it has rotation angle = 90.
        b. If an arbitrary VQA dataset only has a quadrilateral bounding box that has coordinates of an already rotated bounding box => Either it was rotated from bounding box centre or from bounding box bottom corner. **Visualise GT with Jupyter Notebook and don't trust the paper (GeoChat paper said they rotates from corner but only rotation from centre works in GT visualisation)**
        c. If an arbitrary VQA dataset only has a polygon mask, check out online codes to convert polygon to quadrilateral box with and visualise GT along the way.

- (b3) -> (b6) can be done in 1 step using a regular string pattern.
- I might argue to have VQA prompt processing simplified as a skeleton code, and let each individual in the team modify the skeleton code suitable for their datasets. Otherwise if I try to take into account of every dataset. 2 things will happen:
    a. **The script will become very tedious because debugging is like finding a needle in a haystack** (which is the case for HuggingFace). Though any of us can solve this issue, what's better is not starting this potential issue in the first place. On the other hand, if I create the script for every single combined dataset combos like (EarthGPT, GeoChat); (SkyEye, GeoChat) ; (New dataset with new format, GeoChat) then the time taken for to understand is probably 5-10
    b. Data loading will be filled up with lots of these prompt formatting commands and thus making DDP training slower.

TLDR for conclusion: What's better:

- 10-20 minutes coding using GPT4 for every new dataset with different format, and 5-10 minutes debugging a small script.
- 5-10 to investigate how to parse new arguments for every new dataset, at least 30 minutes debugging of a big script if something goes wrong, Dataloader definitely gets slower.