Open in Colab

Note for TA/grader:

After the project draft deadline, I continue working on trying to install the required packages on Google Colab, but I still fail to install the required packages with specific versions on Google Colab. Therefore, in the final report I still use the same strategy to run the code on my own machines with NVIDIA 1080/2080 GPUs, and take a screenshot of my experimental results, then copy the screenshot images to the Google Colab below. To remind you of what problems I have met during the installation on Google Colab, I list them here again:

- The project requires specifically Python 3.7 and pytorch=1.9.1+cu102. However, I tried for a long time to install this specific pytorch version but failed. No matter how I changed the python version or install the older version of pytorch, the environment on Google Colab is always cuda 12.2 with pytorch 2.0.
- I cannot install GLIP and maskrcnn_benchmark on Google Colab, as they need to be manually built (e.g., using python setup.py build develop).

Project Video:

https://drive.google.com/file/d/185NfXDg4ptb8NLxwAG7FuMx0d0ER1hjO/view?usp=sharing

Introduction

- Background of the problem
  - what type of problem: This paper focuses on medical image understanding with pretrained vision language models. The medical image understanding tasks investigated in the paper are mainly referring object detection tasks.
- what is the importance/meaning of solving the problem: Currently vision language models have shown to be extremely powerful in domains like natural images and other image domains, so they have large potential for benefiting the medical image domain.
- what is the difficulty of the problem: The medical image domain requires a very high level of expert knowledge, so it is very challenging to effectively transfer the pretrained knowledge in these vision language models to the medical image domain.
- the state of the art methods and effectiveness: The state-of-the-art solution is to use knowledge transfer techniques to transfer the knowledge from the pretrained vision language models to the specific medical image domain. However, since the medical image domain has much expert knowledge specific in the domain, these knowledge transfer techniques are not that effective.
- Paper explanation
- what did the paper propose: This paper proposes approaches of automatic generation of medical prompts for the pretrained vision language models.
- what is the innovations of the method: The authors conduct a comprehensive study on different strategies of generating the medical prompts, giving a rough guideline of what kinds of prompts are most beneficial for the task.
- how well the proposed method work (in its own metrics): The proposed method outperforms all existing methods in medical image understanding tasks like object detection.
- what is the contribution to the reasearch regime (referring the Background above, how important the paper is to the problem): This paper presents an effective way of generating medical prompts, and successfully utilizing the generated prompts to use the knowledge from pretrained vision language models for medical image understanding. It is the first of accomplishing this goal effectively.

Scope of Reproducibility:

List hypotheses from the paper you will test and the corresponding experiments you will run.

1. Hypothesis 1: Using the Masked Language Model (MLM) technology, we can realize auto-prompt generation.
2. Hypothesis 2: Using the automatically generated prompts, the knowledge transferability from pretrained vision language

model (GLIP used in this paper) to the target medical image domain can be enhanced.

Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

Environment

The required packages are listed in the requirements.txt file which can be referred in the Github repo and also in the original project repo at https://github.com/MembrAI/MIU-VL. The challenging parts of installing the environment include installing the specific version of Pytorch and CUDA. Also, the maskrcnn_benchmark package in the original repo has the inconsistent naming issue as discussed in the project draft.

Data

Data includes raw data (MIMIC III tables), descriptive statistics (our homework questions), and data processing (feature engineering).

- Source of the data: The data is downloaded from https://drive.google.com/file/d/10lSx1yXxfE20nKq6UqquUAD5Egk3hyqi/view?usp=sharing and https://drive.google.com/file/d/1Y2z7FD5p5y31vkZwQQomXFRB0HutHyao/view. Since the officially released dataset (the prior link) is incomplete. The validation data is missing.
- Statistics: include basic descriptive statistics of the dataset like size, cross validation split, label distribution, etc.
- Data process: how do you munipulate the data, e.g., change the class labels, split the dataset to train/valid/test, refining the dataset.
- Illustration: printing results, plotting figures for illustration.

In [ ]:
```python
from maskrcnn_benchmark.data.datasets import CocoDetection
import os
import os.path
import math
from PIL import Image, ImageDraw

import random
import numpy as np

import torch
import torchvision
import torch.utils.data as data
from maskrcnn_benchmark.data.datasets.coco import COCODataset

from maskrcnn_benchmark.structures.bounding_box import BoxList
from maskrcnn_benchmark.structures.segmentation_mask import SegmentationMask
from maskrcnn_benchmark.structures.keypoint import PersonKeypoints
from maskrcnn_benchmark.config import cfg
import pdb
```

In [ ]:
```python
def pil_loader(path, retry=5):
    ri = 0
    while ri < retry:
        try:
            with open(path, 'rb') as f:
                img = Image.open(f)
                return img.convert('RGB')
        except:
            ri += 1
```

In [ ]:
```python
def rgb2id(color):
    if isinstance(color, np.ndarray) and len(color.shape) == 3:
        if color.dtype == np.uint8:
            color = color.astype(np.int32)
        return color[:, :, 0] + 256 * color[:, :, 1] + 256 * 256 * color[:, :, 2]
    return int(color[0] + 256 * color[1] + 256 * 256 * color[2])
```

In [ ]:
```
class CocoDetection(data.Dataset):
    def __init__(self, root, annFile, transform=None, target_transform=None):
        from pycocotools.coco import COCO
        self.root = root
        self.coco = COCO(annFile)
        self.ids = list(self.coco.imgs.keys())
        self.transform = transform
        self.target_transform = target_transform

    def __getitem__(self, index, return_meta=False):
        coco = self.coco
        img_id = self.ids[index]
        if isinstance(img_id, str):
            img_id = [img_id]
        ann_ids = coco.getAnnIds(imgIds=img_id)
        target = coco.loadAnns(ann_ids)

        meta = coco.loadImgs(img_id)[0]
        path = meta['file_name']
        img = pil_loader(os.path.join(self.root, path))

        if self.transform is not None:
            img = self.transform(img)

        if self.target_transform is not None:
            target = self.target_transform(target)

        if return_meta:
            return img, target, meta
        else:
            return img, target, os.path.join(self.root, path)

    def __len__(self):
        return len(self.ids)

    def __repr__(self):
        fmt_str = 'Dataset ' + self.__class__.__name__ + '\n'
        fmt_str += '    Number of datapoints: {}\n'.format(self.__len__())
        fmt_str += '    Root Location: {}\n'.format(self.root)
        tmp = '    Transforms (if any): '
        fmt_str += '{0}{1}\n'.format(tmp, self.transform.__repr__().replace('\n', '\n' + ' ' * len(tmp)))
        tmp = '    Target Transforms (if any): '
        fmt_str += '{0}{1}'.format(tmp, self.target_transform.__repr__().replace('\n', '\n' + ' ' * len(tmp)))
        return fmt_str
```

In [ ]:
```
class VqaCollator(object):
    def __init__(self, size_divisible=0):
        self.size_divisible = size_divisible

    def __call__(self, batch):
        transposed_batch = list(zip(*batch))
        images = transposed_batch[0]
        targets = transposed_batch[1]
        paths = transposed_batch[2]
        return images, targets, paths

def make_dataloader(root, annFile, transforms, **args):
    print(root, annFile, "root!!!!!!!!!!!!!!!")
    dataset = CocoDetection(root, annFile, transforms)
    collate_batch = VqaCollator()
    data_loader = torch.utils.data.DataLoader(
        dataset,
        num_workers=8,
        collate_fn=collate_batch
    )

    return data_loader
```

Model

The model includes the model definitation which usually is a class, model training, and other necessary parts.

- Model architecture: layer number/size/type, activation function, etc
- Training objectives: loss function, optimizer, weight of each loss term, etc
- Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc.

- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

In [ ]:
```
import torch
from torch import nn
import torch.nn.functional as F

from maskrcnn_benchmark.structures.image_list import to_image_list
from maskrcnn_benchmark.structures.bounding_box import BoxList
from maskrcnn_benchmark.structures.boxlist_ops import cat_boxlist

from ..backbone import build_backbone
from ..rpn import build_rpn
from ..roi_heads import build_roi_heads

from ..language_backbone import build_language_backbone
from transformers import AutoTokenizer

import random
import timeit
import pdb
from copy import deepcopy
```

In [ ]:
```
def random_word(input_ids, mask_token_id, vocabs, padding_token_id, greenlight_map):
    output_label = deepcopy(input_ids)
    for j in range(input_ids.size(0)):
        for i in range(input_ids.size(1)):
            prob = random.random()
            ratio = 0.15
            if greenlight_map is not None and greenlight_map[j,i] == -1:
                output_label[j,i] = -100
                continue

            if (not input_ids[j,i] == padding_token_id) and prob < ratio:
                prob /= ratio

                if prob < 0.8:
                    input_ids[j,i] = mask_token_id

                elif prob < 0.9:
                    input_ids[j,i] = random.choice(vocabs)

            else:
                output_label[j,i] = -100

            if greenlight_map is not None and greenlight_map[j,i] != 1:
                output_label[j,i] = -100
    return input_ids, output_label
```

In [ ]:
```
class GeneralizedVLRCNN(nn.Module):
    def __init__(self, cfg):
        super(GeneralizedVLRCNN, self).__init__()
        self.cfg = cfg
        self.backbone = build_backbone(cfg)

        if cfg.MODEL.LANGUAGE_BACKBONE.TOKENIZER_TYPE == "clip":
            from transformers import CLIPTokenizerFast
            if cfg.MODEL.DYHEAD.FUSE_CONFIG.MLM_LOSS:
                print("Reuse token 'ðŁĴij' (token_id = 49404) for mask token!")
                self.tokenizer = CLIPTokenizerFast.from_pretrained("openai/clip-vit-base-patch32",
                                                        from_slow=True, mask_token='ðŁĴij')
            else:
                self.tokenizer = CLIPTokenizerFast.from_pretrained("openai/clip-vit-base-patch32",
                                                        from_slow=True)
        else:
            self.tokenizer = AutoTokenizer.from_pretrained(cfg.MODEL.LANGUAGE_BACKBONE.TOKENIZER_TYPE)
        self.tokenizer_vocab = self.tokenizer.get_vocab()
        self.tokenizer_vocab_ids = [item for key, item in self.tokenizer_vocab.items()]

        self.language_backbone = build_language_backbone(cfg)

        self.rpn = build_rpn(cfg)
        self.roi_heads = build_roi_heads(cfg)
        self.DEBUG = cfg.MODEL.DEBUG

        self.freeze_backbone = cfg.MODEL.BACKBONE.FREEZE
        self.freeze_fpn = cfg.MODEL.FPN.FREEZE
        self.freeze_rpn = cfg.MODEL.RPN.FREEZE
        self.add_linear_layer = cfg.MODEL.DYHEAD.FUSE_CONFIG.ADD_LINEAR_LAYER

        self.force_boxes = cfg.MODEL.RPN.FORCE_BOXES

        if cfg.MODEL.LINEAR_PROB:
            assert cfg.MODEL.BACKBONE.FREEZE, "For linear probing, backbone should be frozen!"
            if hasattr(self.backbone, 'fpn'):
                assert cfg.MODEL.FPN.FREEZE, "For linear probing, FPN should be frozen!"
        self.linear_prob = cfg.MODEL.LINEAR_PROB
        self.freeze_cls_logits = cfg.MODEL.DYHEAD.FUSE_CONFIG.USE_DOT_PRODUCT_TOKEN_LOSS
        if cfg.MODEL.DYHEAD.FUSE_CONFIG.USE_DOT_PRODUCT_TOKEN_LOSS:
            if hasattr(self.rpn.head, 'cls_logits'):
                for p in self.rpn.head.cls_logits.parameters():
                    p.requires_grad = False

        self.freeze_language_backbone = self.cfg.MODEL.LANGUAGE_BACKBONE.FREEZE
        if self.cfg.MODEL.LANGUAGE_BACKBONE.FREEZE:
            for p in self.language_backbone.parameters():
                p.requires_grad = False

        self.use_mlm_loss = cfg.MODEL.DYHEAD.FUSE_CONFIG.MLM_LOSS
        self.mlm_loss_for_only_positives = cfg.MODEL.DYHEAD.FUSE_CONFIG.MLM_LOSS_FOR_ONLY_POSITIVES

    def train(self, mode=True):
        super(GeneralizedVLRCNN, self).train(mode)
        if self.freeze_backbone:
            self.backbone.body.eval()
            for p in self.backbone.body.parameters():
                p.requires_grad = False
        if self.freeze_fpn:
            self.backbone.fpn.eval()
            for p in self.backbone.fpn.parameters():
                p.requires_grad = False
        if self.freeze_rpn:
            if hasattr(self.rpn, 'head'):
                self.rpn.head.eval()
            for p in self.rpn.parameters():
                p.requires_grad = False
        if self.linear_prob:
            if self.rpn is not None:
                for key, value in self.rpn.named_parameters():
                    if not ('bbox_pred' in key or 'cls_logits' in key or 'centerness' in key or 'cosine_scale' in key or '
                        value.requires_grad = False
            if self.roi_heads is not None:
                for key, value in self.roi_heads.named_parameters():
                    if not ('bbox_pred' in key or 'cls_logits' in key or 'centerness' in key or 'cosine_scale' in key or '
                        value.requires_grad = False
        if self.freeze_cls_logits:
            if hasattr(self.rpn.head, 'cls_logits'):
                self.rpn.head.cls_logits.eval()
                for p in self.rpn.head.cls_logits.parameters():
                    p.requires_grad = False
        if self.add_linear_layer:
            if self.rpn is not None:
                for key, p in self.rpn.named_parameters():
                    if 'tunable_linear' in key:
                        p.requires_grad = True

        if self.freeze_language_backbone:
            self.language_backbone.eval()
            for p in self.language_backbone.parameters():
                p.requires_grad = False

    def forward(self,
        images,
        targets=None,
        captions=None,
        positive_map=None,
        greenlight_map=None):
        if self.training and targets is None:
            raise ValueError("In training mode, targets should be passed")
        images = to_image_list(images)
        device = images.tensors.device

        language_dict_features = {}
        if captions is not None:

            tokenized = self.tokenizer.batch_encode_plus(captions,
                                max_length=self.cfg.MODEL.LANGUAGE_BACKBONE.MAX_QUERY_LEN,
                                padding='max_length' if self.cfg.MODEL.LANGUAGE_BACKBONE.PAD_MAX else "longest",
                                    padding="max_length" if self.cfg.MODEL.LANGUAGE_BACKBONE.PAD_MAX else "longest",
                                    return_special_tokens_mask=True,
                                    return_tensors='pt',
                                    truncation=True).to(device)
            if self.use_mlm_loss:
                if not self.mlm_loss_for_only_positives:
                    greenlight_map = None
                input_ids, mlm_labels = random_word(
                    input_ids=tokenized.input_ids,
                    mask_token_id=self.tokenizer.mask_token_id,
                    vocabs=self.tokenizer_vocab_ids,
                    padding_token_id=self.tokenizer.pad_token_id,
                    greenlight_map=greenlight_map)
            else:
                input_ids = tokenized.input_ids
                mlm_labels = None

            tokenizer_input = {"input_ids": input_ids,
                               "attention_mask": tokenized.attention_mask}

            if self.cfg.MODEL.LANGUAGE_BACKBONE.FREEZE:
                with torch.no_grad():
                    language_dict_features = self.language_backbone(tokenizer_input)
            else:
                language_dict_features = self.language_backbone(tokenizer_input)

            if self.cfg.DATASETS.ONE_HOT:
                new_masks = torch.zeros_like(language_dict_features['masks'],
                                             device=language_dict_features['masks'].device)
                new_masks[:, :self.cfg.MODEL.DYHEAD.NUM_CLASSES] = 1
                language_dict_features['masks'] = new_masks

            if self.cfg.MODEL.LANGUAGE_BACKBONE.MASK_SPECIAL:
                language_dict_features["masks"] = 1 - tokenized.special_tokens_mask

            language_dict_features["mlm_labels"] = mlm_labels

        swint_feature_c4 = None
        if 'vl' in self.cfg.MODEL.SWINT.VERSION:
            inputs = {"img": images.tensors, "lang": language_dict_features}
            visual_features, language_dict_features, swint_feature_c4 = self.backbone(inputs)
        else:
            visual_features = self.backbone(images.tensors)

        if targets:
            targets = [target.to(device)
                       for target in targets if target is not None]

        if self.force_boxes:
            proposals = []
            for t in targets:
                tb = t.copy_with_fields(["labels"])
                tb.add_field("scores", torch.ones(tb.bbox.shape[0], dtype=torch.bool, device=tb.bbox.device))
                proposals.append(tb)
            if self.cfg.MODEL.RPN.RETURN_FUSED_FEATURES:
                _, proposal_losses, fused_visual_features = self.rpn(
                    images, visual_features, targets, language_dict_features,
                    positive_map, captions, swint_feature_c4)
            elif self.training:
                null_loss = 0
                for key, param in self.rpn.named_parameters():
                    null_loss += 0.0 * param.sum()
                proposal_losses = {('rpn_null_loss', null_loss)}
        else:
            proposals, proposal_losses, fused_visual_features = self.rpn(images, visual_features, targets, language_dict_f
                                              captions, swint_feature_c4)
        if self.roi_heads:
            if self.cfg.MODEL.ROI_MASK_HEAD.PREDICTOR.startswith("VL"):
                if self.training:
                    assert len(targets) == 1 and len(targets[0]) == len(positive_map), "shape match assert for mask head!!
                    targets[0].add_field("positive_map", positive_map)
            if self.cfg.MODEL.RPN.RETURN_FUSED_FEATURES:
                x, result, detector_losses = self.roi_heads(
                    fused_visual_features, proposals, targets,
                    language_dict_features=language_dict_features,
                    positive_map_label_to_token=positive_map if not self.training else None
                )
            else:
                x, result, detector_losses = self.roi_heads(
                    visual_features, proposals, targets,
                    language_dict_features=language_dict_features,
                positive_map_label_to_token=positive_map if not self.training else None
            )
        else:
            x = visual_features
            result = proposals
            detector_losses = {}

        if self.training:
            losses = {}
            losses.update(detector_losses)
            losses.update(proposal_losses)
            return losses

        return result
```

Training & Evaluation

This paper is focusing on leveraging pretrained vision-language models (VLMs) for medical image understanding. Therefore, it focuses on how to utilize the prompt to adapt the pretrained VLMs to the new medical image domain. Therefore, this model can do zero-shot medical image understanding tasks without training. The evaluation code is shown as below:

In [ ]:

```
def evaluate(self):
    """
    Run per image evaluation on given images and store results
    (a list of dict) in self.eval_imgs.
    """

    self.params.img_ids = list(np.unique(self.params.img_ids))

    if self.params.use_cats:
        cat_ids = self.params.cat_ids
    else:
        cat_ids = [-1]

    self._prepare()

    self.ious = {
        (img_id, cat_id): self.compute_iou(img_id, cat_id) for img_id in self.params.img_ids for cat_id in cat_ids
    }

    # Loop through images, area range, max detection number
    self.eval_imgs = [
        self.evaluate_img(img_id, cat_id, area_rng)
        for cat_id in cat_ids
        for area_rng in self.params.area_rng
        for img_id in self.params.img_ids
    ]

def _get_gt_dt(self, img_id, cat_id):
    """Create gt, dt which are list of anns/dets. If use_cats is true
    only anns/dets corresponding to tuple (img_id, cat_id) will be
    used. Else, all anns/dets in image are used and cat_id is not used.
    """
    if self.params.use_cats:
        gt = self._gts[img_id, cat_id]
        dt = self._dts[img_id, cat_id]
    else:
        gt = [_ann for _cat_id in self.params.cat_ids for _ann in self._gts[img_id, cat_id]]
        dt = [_ann for _cat_id in self.params.cat_ids for _ann in self._dts[img_id, cat_id]]
    return gt, dt

def compute_iou(self, img_id, cat_id):
    gt, dt = self._get_gt_dt(img_id, cat_id)

    if len(gt) == 0 and len(dt) == 0:
        return []

    idx = np.argsort([-d["score"] for d in dt], kind="mergesort")
    dt = [dt[i] for i in idx]

    iscrowd = [int(False)] * len(gt)

    if self.params.iou_type == "segm":
        ann_type = "segmentation"
    elif self.params.iou_type == "bbox":
        ann_type = "bbox"
        else:
            raise ValueError("Unknown iou_type for iou computation.")
        gt = [g[ann_type] for g in gt]
        dt = [d[ann_type] for d in dt]

        ious = mask_util.iou(dt, gt, iscrowd)
        return ious

    def evaluate_img(self, img_id, cat_id, area_rng):
        """Perform evaluation for single category and image."""
        gt, dt = self._get_gt_dt(img_id, cat_id)

        if len(gt) == 0 and len(dt) == 0:
            return None

        for g in gt:
            if g["ignore"] or (g["area"] < area_rng[0] or g["area"] > area_rng[1]):
                g["_ignore"] = 1
            else:
                g["_ignore"] = 0

        gt_idx = np.argsort([g["_ignore"] for g in gt], kind="mergesort")
        gt = [gt[i] for i in gt_idx]

        dt_idx = np.argsort([-d["score"] for d in dt], kind="mergesort")
        dt = [dt[i] for i in dt_idx]
        ious = self.ious[img_id, cat_id][:, gt_idx] if len(self.ious[img_id, cat_id]) > 0 else self.ious[img_id, cat_id]

        num_thrs = len(self.params.iou_thrs)
        num_gt = len(gt)
        num_dt = len(dt)

        gt_m = np.zeros((num_thrs, num_gt))
        dt_m = np.zeros((num_thrs, num_dt))

        gt_ig = np.array([g["_ignore"] for g in gt])
        dt_ig = np.zeros((num_thrs, num_dt))

        for iou_thr_idx, iou_thr in enumerate(self.params.iou_thrs):
            if len(ious) == 0:
                break

            for dt_idx, _dt in enumerate(dt):
                iou = min([iou_thr, 1 - 1e-10])

                m = -1
                for gt_idx, _ in enumerate(gt):
                    if gt_m[iou_thr_idx, gt_idx] > 0:
                        continue
                    if m > -1 and gt_ig[m] == 0 and gt_ig[gt_idx] == 1:
                        break
                    if ious[dt_idx, gt_idx] < iou:
                        continue
                    iou = ious[dt_idx, gt_idx]
                    m = gt_idx

                if m == -1:
                    continue


                dt_ig[iou_thr_idx, dt_idx] = gt_ig[m]
                dt_m[iou_thr_idx, dt_idx] = gt[m]["id"]
                gt_m[iou_thr_idx, m] = _dt["id"]


        dt_ig_mask = [
            d["area"] < area_rng[0] or d["area"] > area_rng[1] or d["category_id"] in self.img_nel[d["image_id"]]
            for d in dt
        ]
        dt_ig_mask = np.array(dt_ig_mask).reshape((1, num_dt))
        dt_ig_mask = np.repeat(dt_ig_mask, num_thrs, 0)
        dt_ig = np.logical_or(dt_ig, np.logical_and(dt_m == 0, dt_ig_mask))
        return {
            "image_id": img_id,
            "category_id": cat_id,
            "area_rng": area_rng,
            "dt_ids": [d["id"] for d in dt],
            "gt_ids": [g["id"] for g in gt],
            "dt_matches": dt_m,
            "gt_matches": gt_m,
            "dt_scores": [d["score"] for d in dt],
        "dt_scores": [d['score'] for d in dt],
        "gt_ignore": gt_ig,
        "dt_ignore": dt_ig,
    }
```

Evaluation Metrics Explanation

The evaluation metric is mainly mAP and mIOU for polyp detection. mAP is the average precision which combines recall and precision for ranked retrieval results. mIOU means "mean intersection over union", which is a metric prevalently used in segmentation and detection. The intersection means the intersection region of the predicted mask and the ground truth mask, while the union means the union region of the predicted mask and the ground truth mask. Therefore, IOU is the value of intersection divided by the value of union, and mIOU is IOU taking the average over all classes.

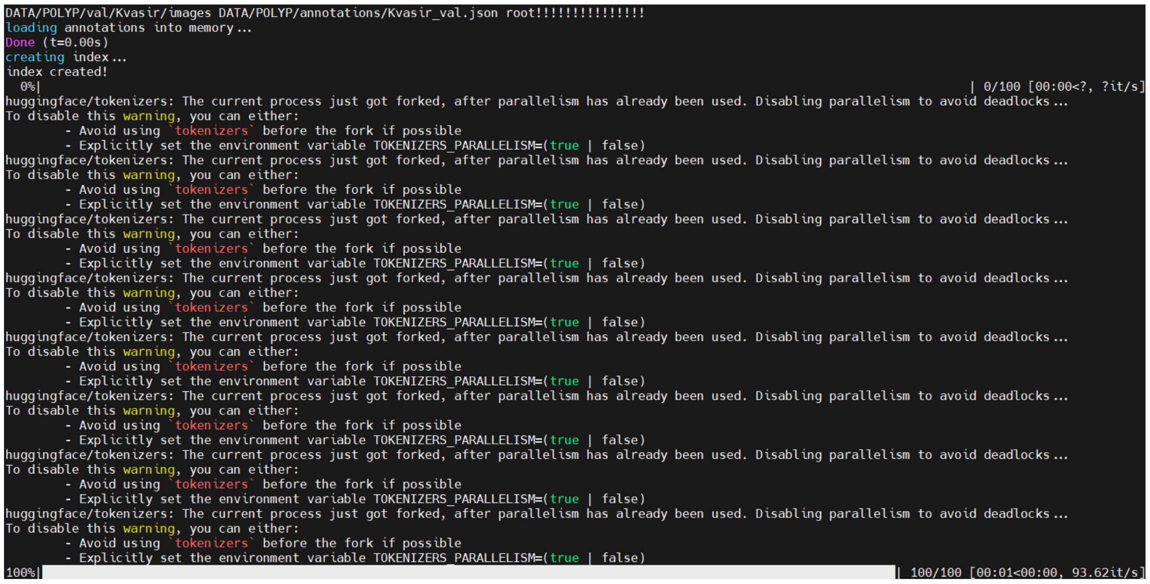
Results

In this section, you should finish training your model training or loading your trained model. That is a great experiment! You should share the results with others with necessary metrics and figures.

Please test and report results for all experiments that you run with:

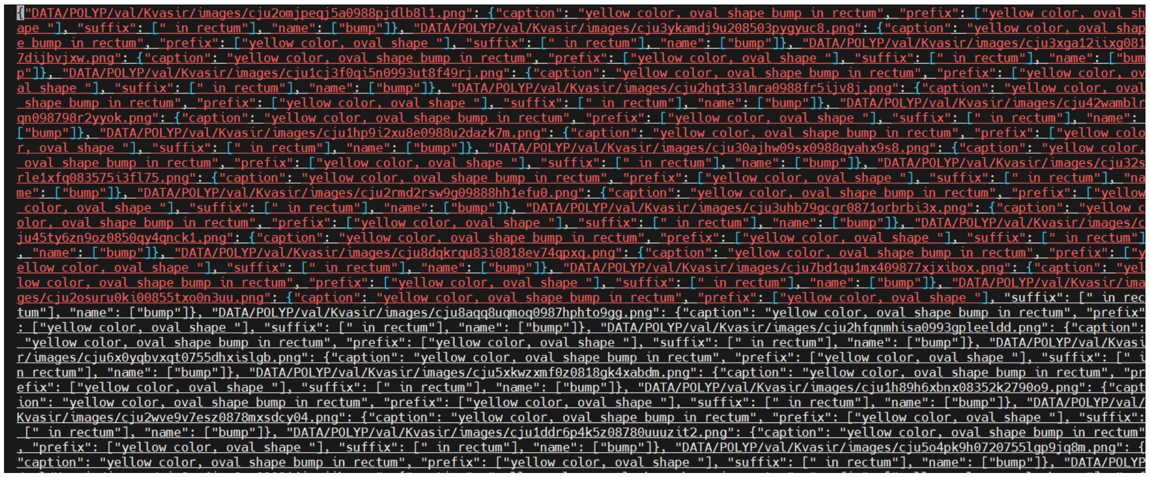
- specific numbers (accuracy, AUC, RMSE, etc)
- figures (loss shrinkage, outputs from GAN, annotation or label of sample pictures, etc)

This is the screenshot of using Masked Language Modeling to generate prompts automatically:

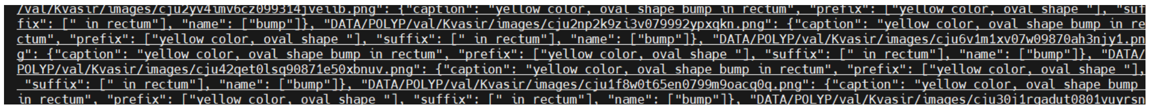


```
DATA/POLYP/val/Kvasir/images DATA/POLYP/annotations/Kvasir_val.json root!!!!!!!!!!!!!!!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
  0%|                                                                                          | 0/100 [00:00<?, ?it/s]
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
100%|██████████████████████████████████████████████████████████████████████████████████| 100/100 [00:01<00:00, 93.62it/s]
```

These are some samples of the generated prompts:



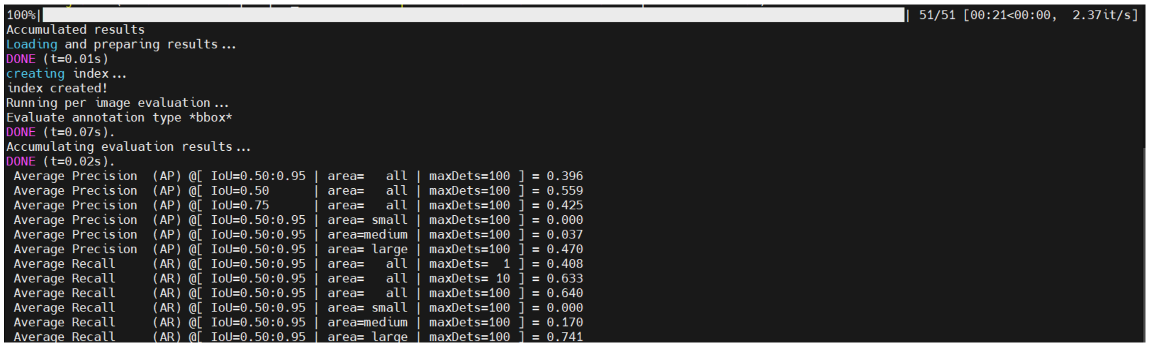
```
{"DATA/POLYP/val/Kvasir/images/cju2omjpeqj5a0988pjdlb8l1.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval sh
ape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju3ykamdj9u208503pygyuc8.png": {"caption": "yellow color, oval shap
e bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju3xga12iixg081
7dijbvjxw.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bum
p"]}, "DATA/POLYP/val/Kvasir/images/cju1cj3f0qi5n0993ut8f49rj.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, ov
al shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju2hqt33lmra0988fr5ijv8j.png": {"caption": "yellow color, oval
 shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju42wamblr
qn098798r2yyok.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": 
["bump"]}, "DATA/POLYP/val/Kvasir/images/cju1hp9i2xu8e0988u2dazk7m.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow colo
r, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju30ajhw09sx0988qyahx9s8.png": {"caption": "yellow color,
 oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju32s
rle1xfq083575i3fl75.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "na
me": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju2rmd2rsw9g09888hh1efu0.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow
 color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju3uhb79gcgr0871orbrbi3x.png": {"caption": "yellow c
olor, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/c
ju45ty6zn9oz0850qy4qnck1.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"]
, "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju8dqkrqu83i0818ev74qpxq.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["y
ellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju7bd1qu1mx409877xjxibox.png": {"caption": "yel
low color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/ima
ges/cju2osuru0ki00855txo0n3uu.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rec
tum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju8aqq8uqmoq0987hphto9gg.png": {"caption": "yellow color, oval shape bump in rectum", "prefix"
: ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju2hfqnmhisa0993gpleeldd.png": {"caption":
 "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasi
r/images/cju6x0yqbvxqt0755dhxislgb.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" i
n rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju5xkwzxmf0z0818gk4xabdm.png": {"caption": "yellow color, oval shape bump in rectum", "pr
efix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju1h89h6xbnx08352k2790o9.png": {"capt
ion": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/
Kvasir/images/cju2wve9v7esz0878mxsdcy04.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix":
 [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju1ddr6p4k5z08780uuuzit2.png": {"caption": "yellow color, oval shape bump in rectum"
, "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju5o4pk9h0720755lgp9jq8m.png": {
"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP
```



```
/val/Kvasir/images/cju2yv4imv6cz099314jveiib.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suf
fix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju2np2k9zi3v079992ypxqkn.png": {"caption": "yellow color, oval shape bump in re
ctum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju6v1m1xv07w09870ah3njy1.pn
g": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/
POLYP/val/Kvasir/images/cju42qet0lsq90871e50xbnuv.png": {"caption": "yellow color, oval shape bump in rectum", "prefix": ["yellow color, oval shape "],
 "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju1f8w0t65en0799m9oacq0q.png": {"caption": "yellow color, oval shape bump
 in rectum", "prefix": ["yellow color, oval shape "], "suffix": [" in rectum"], "name": ["bump"]}, "DATA/POLYP/val/Kvasir/images/cju30j1rgadut0801vuyrsn
```

We can see that they are in the form of "xxx color, xxx shape xxx in xxx".

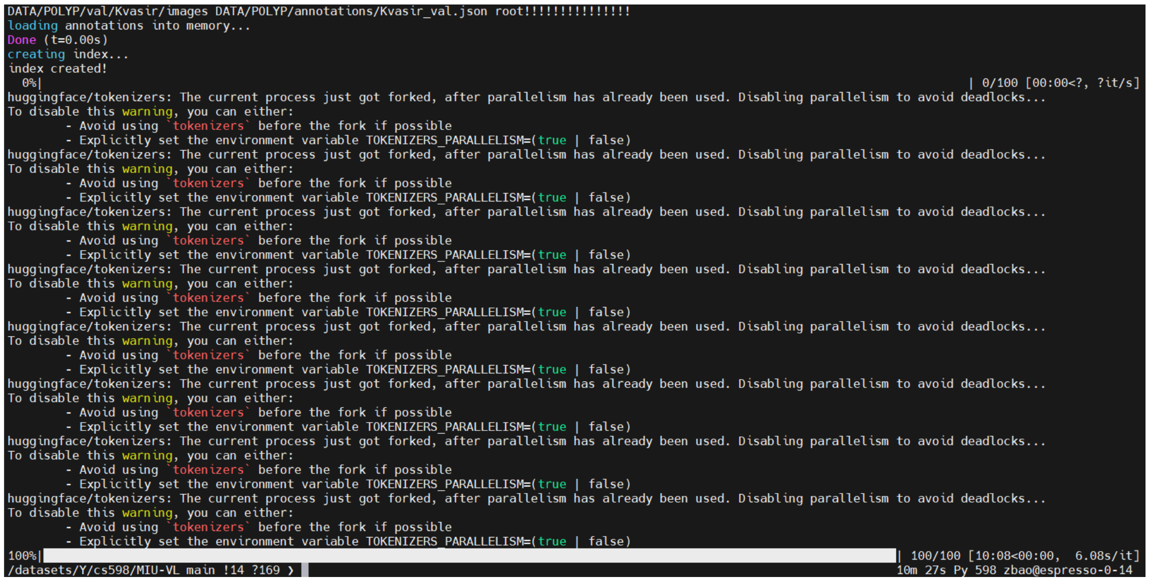
This is the result of completely using auto-generated prompts by Masked Language Modeling (MLM):



```
100%|████████████████████████████████████████████████████████████████████| 51/51 [00:21<00:00,  2.37it/s]
Accumulated results
Loading and preparing results...
DONE (t=0.01s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.07s).
Accumulating evaluation results...
DONE (t=0.02s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.396
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.559
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.425
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.037
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.470
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.408
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.633
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.640
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.170
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.741
```

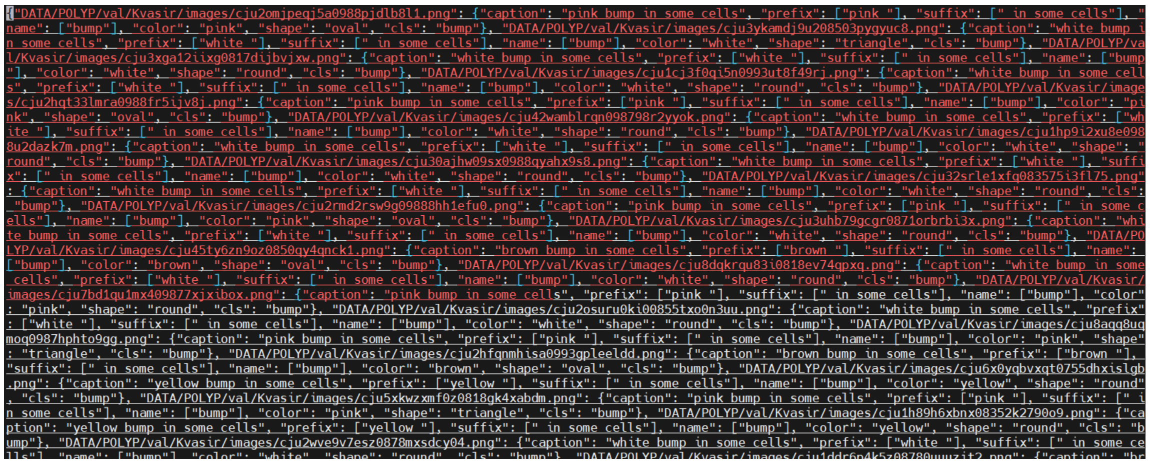
So after the project draft deadline, I have made the OFA model work so that we could run the results for the hybrid strategy of generating the prompts.

This is the screenshot of using hybrid method to generate prompts automatically:

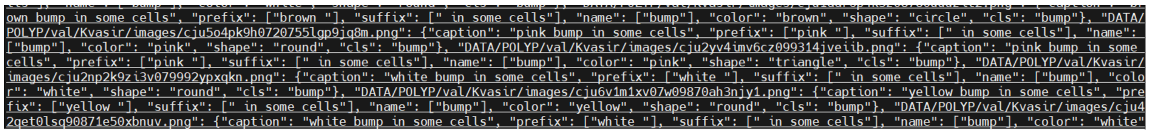


```
DATA/POLYP/val/Kvasir/images DATA/POLYP/annotations/Kvasir_val.json root!!!!!!!!!!!!!!!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
  0%|                                                                                  | 0/100 [00:00<?, ?it/s]
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
100%|████████████████████████████████████████████████████████████████████| 100/100 [10:08<00:00,  6.08s/it]
/datasets/Y/cs598/MIU-VL main !14 ?169 ❯                                       10m 27s Py 598 zbao@espresso-0-14
```

These are some samples of the generated prompts:



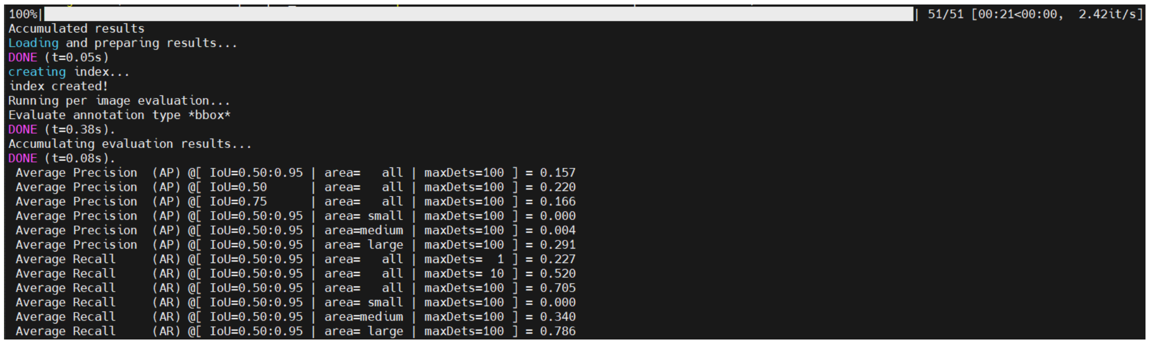
```
{"DATA/POLYP/val/Kvasir/images/cju2omjpeqj5a0988pjdlb8l1.png": {"caption": "pink bump in some cells", "prefix": ["pink "], "suffix": [" in some cells"], "
name": ["bump"], "color": "pink", "shape": "oval", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju3ykamdj9u208503pygyuc8.png": {"caption": "white bump i
n some cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "triangle", "cls": "bump"}, "DATA/POLYP/va
l/Kvasir/images/cju3xga12iixg0817dijbvjxw.png": {"caption": "white bump in some cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump
"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju1cj3f0qi5n0993ut8f49rj.png": {"caption": "white bump in some cell
s", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/image
s/cju2hqt33lmra0988fr5ijv8j.png": {"caption": "pink bump in some cells", "prefix": ["pink "], "suffix": [" in some cells"], "name": ["bump"], "color": "pi
nk", "shape": "oval", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju42wamblrqn098798r2yyok.png": {"caption": "white bump in some cells", "prefix": ["wh
ite "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju1hp9i2xu8e098
8u2dazk7m.png": {"caption": "white bump in some cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "
round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju30ajhw09sx0988qyahx9s8.png": {"caption": "white bump in some cells", "prefix": ["white "], "suffi
x": [" in some cells"], "name": ["bump"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju32srle1xfq083575i3fl75.png"
: {"caption": "white bump in some cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "round", "cls":
 "bump"}, "DATA/POLYP/val/Kvasir/images/cju2rmd2rsw9g09888hh1efu0.png": {"caption": "pink bump in some cells", "prefix": ["pink "], "suffix": [" in some c
ells"], "name": ["bump"], "color": "pink", "shape": "oval", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju3uhb79gcgr0871orbrbi3x.png": {"caption": "whi
te bump in some cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/PO
LYP/val/Kvasir/images/cju45ty6zn9oz0850qy4qnck1.png": {"caption": "brown bump in some cells", "prefix": ["brown "], "suffix": [" in some cells"], "name":
["bump"], "color": "brown", "shape": "oval", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju8dqkrqu83i0818ev74qpxq.png": {"caption": "white bump in some
 cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/
images/cju7bd1qu1mx409877xjxibox.png": {"caption": "pink bump in some cells", "prefix": ["pink "], "suffix": [" in some cells"], "name": ["bump"], "color"
: "pink", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju2osuru0ki00855txo0n3uu.png": {"caption": "white bump in some cells", "prefix"
: ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju8aqq8uq
moq0987hphto9gg.png": {"caption": "pink bump in some cells", "prefix": ["pink "], "suffix": [" in some cells"], "name": ["bump"], "color": "pink", "shape"
: "triangle", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju2hfqnmhisa0993gpleeldd.png": {"caption": "brown bump in some cells", "prefix": ["brown "],
"suffix": [" in some cells"], "name": ["bump"], "color": "brown", "shape": "oval", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju6x0yqbvxqt0755dhxislgb
.png": {"caption": "yellow bump in some cells", "prefix": ["yellow "], "suffix": [" in some cells"], "name": ["bump"], "color": "yellow", "shape": "round"
, "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju5xkwzxmf0z0818gk4xabdm.png": {"caption": "pink bump in some cells", "prefix": ["pink "], "suffix": [" i
n some cells"], "name": ["bump"], "color": "pink", "shape": "triangle", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju1h89h6xbnx08352k279009.png": {"ca
ption": "yellow bump in some cells", "prefix": ["yellow "], "suffix": [" in some cells"], "name": ["bump"], "color": "yellow", "shape": "round", "cls": "b
ump"}, "DATA/POLYP/val/Kvasir/images/cju2wve9v7esz0878mxsdcy04.png": {"caption": "white bump in some cells", "prefix": ["white "], "suffix": [" in some ce
lls"], "name": ["bump"], "color": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju1ddr6p4k5z08780uuuzit2.png": {"caption": "br
```



```
own bump in some cells", "prefix": ["brown "], "suffix": [" in some cells"], "name": ["bump"], "color": "brown", "shape": "circle", "cls": "bump"}, "DATA/
POLYP/val/Kvasir/images/cju5o4pk9h0720755lgp9jq8m.png": {"caption": "pink bump in some cells", "prefix": ["pink "], "suffix": [" in some cells"], "name":
["bump"], "color": "pink", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju2yv4imv6cz099314jveiib.png": {"caption": "pink bump in some
cells", "prefix": ["pink "], "suffix": [" in some cells"], "name": ["bump"], "color": "pink", "shape": "triangle", "cls": "bump"}, "DATA/POLYP/val/Kvasir/
images/cju2np2k9zi3v079992ypxqkn.png": {"caption": "white bump in some cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "colo
r": "white", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju6v1m1xv07w09870ah3njy1.png": {"caption": "yellow bump in some cells", "pre
fix": ["yellow "], "suffix": [" in some cells"], "name": ["bump"], "color": "yellow", "shape": "round", "cls": "bump"}, "DATA/POLYP/val/Kvasir/images/cju4
2qet0lsq90871e50xbnuv.png": {"caption": "white bump in some cells", "prefix": ["white "], "suffix": [" in some cells"], "name": ["bump"], "color": "white"
```

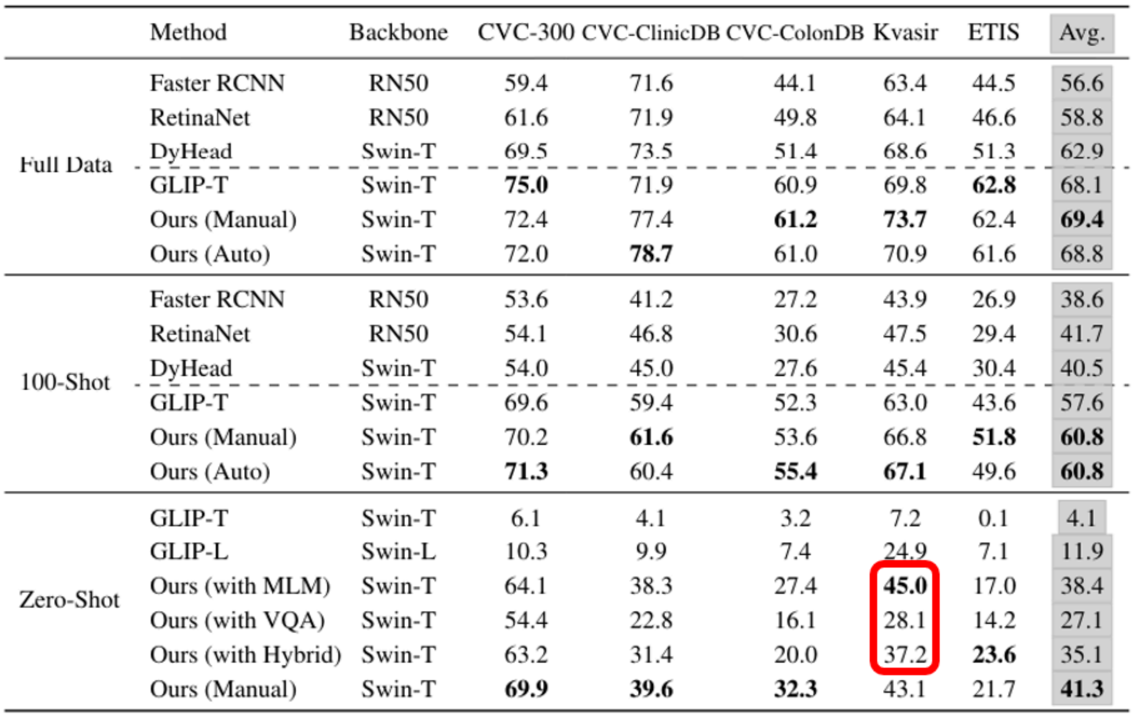
We can see that they are in the form of "xxx bump in some cells", where the prompts are less flexible and informative than masked image modeling (MLM) strategy.

This is the result of completely using auto-generated prompts by the hybrid method:



```
100%|████████████████████████████████████████████████████████████| 51/51 [00:21<00:00,  2.42it/s]
Accumulated results
Loading and preparing results...
DONE (t=0.05s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.38s).
Accumulating evaluation results...
DONE (t=0.08s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.157
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.220
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.166
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.004
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.291
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.227
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.520
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.705
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.340
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.786
```

When comparing with the results in the paper, we can see that the results are actually close to the results of the Kvasir dataset in Table 3, as shown in the red box region as follows:



	Method	Backbone	CVC-300	CVC-ClinicDB	CVC-ColonDB	Kvasir	ETIS	Avg.
Full Data	Faster RCNN	RN50	59.4	71.6	44.1	63.4	44.5	56.6
	RetinaNet	RN50	61.6	71.9	49.8	64.1	46.6	58.8
	DyHead	Swin-T	69.5	73.5	51.4	68.6	51.3	62.9
	GLIP-T	Swin-T	**75.0**	71.9	60.9	69.8	**62.8**	68.1
	Ours (Manual)	Swin-T	72.4	77.4	**61.2**	**73.7**	62.4	**69.4**
	Ours (Auto)	Swin-T	72.0	**78.7**	61.0	70.9	61.6	68.8
100-Shot	Faster RCNN	RN50	53.6	41.2	27.2	43.9	26.9	38.6
	RetinaNet	RN50	54.1	46.8	30.6	47.5	29.4	41.7
	DyHead	Swin-T	54.0	45.0	27.6	45.4	30.4	40.5
	GLIP-T	Swin-T	69.6	59.4	52.3	63.0	43.6	57.6
	Ours (Manual)	Swin-T	70.2	**61.6**	53.6	66.8	**51.8**	**60.8**
	Ours (Auto)	Swin-T	**71.3**	60.4	**55.4**	**67.1**	49.6	**60.8**
Zero-Shot	GLIP-T	Swin-T	6.1	4.1	3.2	7.2	0.1	4.1
	GLIP-L	Swin-L	10.3	9.9	7.4	24.9	7.1	11.9
	Ours (with MLM)	Swin-T	64.1	38.3	27.4	**45.0**	17.0	38.4
	Ours (with VQA)	Swin-T	54.4	22.8	16.1	28.1	14.2	27.1
	Ours (with Hybrid)	Swin-T	63.2	31.4	20.0	37.2	**23.6**	35.1
	Ours (Manual)	Swin-T	**69.9**	**39.6**	**32.3**	43.1	21.7	**41.3**

Also, the trend of MLM outperforms the hybrid method also conforms with the experimental results conducted by me, as from the above results we can see that MLM generally has better performance than hybrid method. On the average precision part the superiority is obvious although for the average recall the hybrid method performs slightly better. Nevertheless, the average recall metric is not reported in the paper.

Discussion

In this section,you should discuss your work and make future plan. The discussion should address the following questions:

- Explain why it is not reproducible if your results are kind negative.

- Describe “What was easy” and “What was difficult” during the reproduction.
- Make suggestions to the author or other reproducers on how to improve the reproducibility.
- I think this paper is reproducible. As discussed above, we could achieve comparable results as the paper reports. Also, our observation from the results of generating prompts with MLM is better than hybrid method is also consistent with the results in the paper.
- The coding part of this project relatively easy, because the authors have provided the official implementation on GitHub. However, the environment building part is very difficult for me, as there are many dependencies and requirements.
- I would recommend authors could try not using some packages that has specific version requirements, which makes it hard for other researchers to install the environment and reproduce the results.

Public GitHub Repo

https://github.com/zsh2000/CS598-DL4H

References

1. Qin et al. Medical Image Understanding with Pretrained Vision Language Models: A Comprehensive Study. ICLR 2023.