```
!git clone https://github.com/Heidelberg-NLP/MM-SHAP.git
```

```
Cloning into 'MM-SHAP'...
remote: Enumerating objects: 166, done.
remote: Counting objects: 100% (166/166), done.
remote: Compressing objects: 100% (148/148), done.
remote: Total 166 (delta 20), reused 149 (delta 12), pack-reused 0 (from 0)
Receiving objects: 100% (166/166), 459.05 KiB | 2.49 MiB/s, done.
Resolving deltas: 100% (20/20), done.
```

```
#Colour Coding is done to show the representation of each token in predicting the output on the basis of their colour.
#Light color mean negative contribution and Dark Color means positive contribution
```

```
# conda activate shap (rampage)
import shap
import torch
import sklearn
from torch import nn
from torchvision import transforms
from PIL import Image
import numpy as np
import os, copy, json
import re, math, sys
import random
from tqdm import tqdm
from functools import partial
sys.path.insert(0, 'MM-SHAP') # Add MM-SHAP directory to Python path

from PIL import Image
from transformers import Qwen2_5_VLForConditionalGeneration, AutoProcessor, CLIPImageProcessor
```

```
import pandas as pd
df = pd.read_csv('us_test.csv', on_bad_lines='skip')
df = df.iloc[:5]
df = df[['id' , 'summaries' ,'verdict']]
print(df)
```

```
       id                                          summaries verdict
0   42033  13. Plaintiff is a natural person allegedly ob...     win
1  424269  Beginning in or around October <DATE>, Defenda...    lose
2  264236  Demark is a professional consulting and engine...     win
3   74553  Defendant is a swimwear company that owns and ...    lose
4  113054  Plaintiff brings this action on behalf of hers...    lose
```

```
num_samples = "all"
if num_samples != "all":
    num_samples = int(num_samples)

write_res = "yes"
task = "image_sentence_alignment"
other_tasks_than_valse = ['mscoco','vqa','gqa','gqa_balanced','nlvr2']
use_cuda = True

model_name = "Qwen/Qwen2.5-VL-3B-Instruct"  # or other variant

model = Qwen2_5_VLForConditionalGeneration.from_pretrained(
        model_name,
        dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
        device_map="auto" if torch.cuda.is_available() else None,
        trust_remote_code=True,

    )

processor = AutoProcessor.from_pretrained(model_name, trust_remote_code=True)
print(processor.__dict__ )
model.eval()
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json:         1.37k/? [00:00<00:00, 117kB/s]

model.safetensors.index.json:         65.4k/? [00:00<00:00, 4.66MB/s]

Fetching 2 files: 100%                                                  2/2 [01:08<00:00, 28.54s/it]

model-00002-of-00002.safetensors: 100%                                  3.53G/3.53G [01:08<00:00, 211MB/s]

model-00001-of-00002.safetensors: 100%                                  3.98G/3.98G [01:06<00:00, 92.0MB/s]

Loading checkpoint shards: 100%                                         2/2 [00:23<00:00, 12.29s/it]

generation_config.json: 100%                                            216/216 [00:00<00:00, 25.1kB/s]

preprocessor_config.json: 100%                                          350/350 [00:00<00:00, 44.2kB/s]

The image processor of type `Qwen2VLImageProcessor` is now loaded as a fast processor by default, even if the model checkpoi
tokenizer_config.json:         5.70k/? [00:00<00:00, 601kB/s]

vocab.json:         2.78M/? [00:00<00:00, 45.3MB/s]

merges.txt:         1.67M/? [00:00<00:00, 50.4MB/s]

tokenizer.json:         7.03M/? [00:00<00:00, 110MB/s]

chat_template.json:         1.05k/? [00:00<00:00, 116kB/s]

{'image_token': '<|image_pad|>', 'video_token': '<|video_pad|>', 'image_token_id': 151655, 'video_token_id': 151656, 'chat_t
  "crop_size": null,
  "data_format": "channels_first",
  "default_to_square": true,
  "device": null,
  "disable_grouping": null,
  "do_center_crop": null,
  "do_convert_rgb": true,
  "do_normalize": true,
  "do_pad": null,
  "do_rescale": true,
  "do_resize": true,
  "image_mean": [
    0.48145466,
    0.4578275,
    0.40821073
  ],
  "image_processor_type": "Qwen2VLImageProcessorFast",
  "image_std": [
    0.26862954,
    0.26130258,
    0.27577711
  ],
  "input_data_format": null,
  "max_pixels": 12845056,
  "merge_size": 2,
  "min_pixels": 3136,
  "pad_size": null,
  "patch_size": 14,
  "processor_class": "Qwen2_5_VLProcessor",
  "resample": 3,
  "rescale_factor": 0.00392156862745098,
  "return_tensors": null,
  "size": {
    "longest_edge": 12845056,
    "shortest_edge": 3136
  },
  "temporal_patch_size": 2
}
, 'tokenizer': Qwen2TokenizerFast(name_or_path='Qwen/Qwen2.5-VL-3B-Instruct', vocab_size=151643, model_max_length=131072, is
    151643: AddedToken("<|endoftext|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151644: AddedToken("<|im_start|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151645: AddedToken("<|im_end|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151646: AddedToken("<|object_ref_start|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True
    151647: AddedToken("<|object_ref_end|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151648: AddedToken("<|box_start|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151649: AddedToken("<|box_end|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151650: AddedToken("<|quad_start|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151651: AddedToken("<|quad_end|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151652: AddedToken("<|vision_start|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151653: AddedToken("<|vision_end|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151654: AddedToken("<|vision_pad|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151655: AddedToken("<|image_pad|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151656: AddedToken("<|video_pad|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
    151657: AddedToken("<tool_call>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
    151658: AddedToken("</tool_call>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
    151659: AddedToken("<|fim_prefix|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
    151660: AddedToken("<|fim_middle|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
    151661: AddedToken("<|fim_suffix|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
```

```
    151662: AddedToken("<|fim_pad|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
    151663: AddedToken("<|repo_name|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
    151664: AddedToken("<|file_sep|>", rstrip=False, lstrip=False, single_word=False, normalized=False, special=False),
}
), 'video_processor': Qwen2VLVideoProcessor {
  "crop_size": null,
  "data_format": "channels_first",
  "default_to_square": true,
  "device": null,
  "do_center_crop": null,
  "do_convert_rgb": true,
  "do_normalize": true,
  "do_rescale": true,
  "do_resize": true,
  "do_sample_frames": false,
  "fps": null,
  "image_mean": [
    0.48145466,
    0.4578275,
    0.40821073
  ],
  "image_std": [
    0.26862954,
    0.26130258,
    0.27577711
  ],
  "input_data_format": null,
  "max_frames": 768,
  "max_pixels": 12845056,
  "merge_size": 2,
  "min_frames": 4,
  "min_pixels": 3136,
  "num_frames": null,
  "pad_size": null,
  "patch_size": 14,
  "processor_class": "Qwen2_5_VLProcessor",
  "resample": 3,
  "rescale_factor": 0.00392156862745098,
  "return_metadata": false,
  "size": {
    "longest_edge": 12845056,
    "shortest_edge": 3136
  },
  "temporal_patch_size": 2,
  "video_metadata": null,
  "video_processor_type": "Qwen2VLVideoProcessor"
}
}
Qwen2_5_VLForConditionalGeneration(
  (model): Qwen2_5_VLModel(
    (visual): Qwen2_5_VisionTransformerPretrainedModel(
      (patch_embed): Qwen2_5_VisionPatchEmbed(
        (proj): Conv3d(3, 1280, kernel_size=(2, 14, 14), stride=(2, 14, 14), bias=False)
      )
      (rotary_pos_emb): Qwen2_5_VisionRotaryEmbedding()
      (blocks): ModuleList(
        (0-31): 32 x Qwen2_5_VLVisionBlock(
          (norm1): Qwen2RMSNorm((1280,), eps=1e-06)
          (norm2): Qwen2RMSNorm((1280,), eps=1e-06)
          (attn): Qwen2_5_VLVisionAttention(
            (qkv): Linear(in_features=1280, out_features=3840, bias=True)
            (proj): Linear(in_features=1280, out_features=1280, bias=True)
          )
          (mlp): Qwen2_5_VLMLP(
            (gate_proj): Linear(in_features=1280, out_features=3420, bias=True)
            (up_proj): Linear(in_features=1280, out_features=3420, bias=True)
            (down_proj): Linear(in_features=3420, out_features=1280, bias=True)
            (act_fn): SiLUActivation()
          )
        )
      )
      (merger): Qwen2_5_VLPatchMerger(
        (ln_q): Qwen2RMSNorm((1280,), eps=1e-06)
        (mlp): Sequential(
          (0): Linear(in_features=5120, out_features=5120, bias=True)
          (1): GELU(approximate='none')
          (2): Linear(in_features=5120, out_features=2048, bias=True)
        )
      )
    )
    (language_model): Qwen2_5_VLTextModel(
      (embed_tokens): Embedding(151936, 2048)
      (layers): ModuleList(
        (0-35): 36 x Qwen2_5_VLDecoderLayer(
          (self_attn): Qwen2_5_VLAttention(
            (q_proj): Linear(in_features=2048, out_features=2048, bias=True)
            (k_proj): Linear(in_features=2048, out_features=256, bias=True)
            (v_proj): Linear(in_features=2048, out_features=256, bias=True)
            (o_proj): Linear(in_features=2048, out_features=2048, bias=False)
            (rotary_emb): Qwen2_5_VLRotaryEmbedding()
          )
```

```
def custom_masker(mask, x):
    """
    Shap relevant function.
    It gets a mask from the shap library with truth values about which image and text tokens to mask (False) and which not
    It defines how to mask the text tokens and masks the text tokens. So far, we don't mask the image, but have only define
    """
    masked_X = x.clone()
    mask = torch.tensor(mask).unsqueeze(0)
    masked_X[~mask] = 0  # ~mask !!! to zero
    for i in range(masked_X.shape[0]):
      for j in range(masked_X.shape[1]):
       if x[i,j] >= 151643 and x[i,j]<=151664 :
         masked_X[i,j]=x[i,j]
    return masked_X
def compute_mm_score(text_length, shap_values):
    """ Compute Multimodality Score. (80% textual, 20% visual, possibly: 0% knowledge). """
    sum=0
    for i in range(text_length):
      if inputs["input_ids"][0][i] >= 151643 and inputs["input_ids"][0][i]<=151664 :
        continue
      else:
        sum = sum + np.abs(shap_values.values[0, 0, i])
    text_contrib = sum
    image_contrib = np.abs(shap_values.values[0, 0, text_length:]).sum()
    text_score = text_contrib / (text_contrib + image_contrib)
    # image_score = image_contrib / (text_contrib + image_contrib) # is just 1 - text_score in the two modalities case
    return text_score
```

```
def get_model_prediction(x):
    """
    Shap relevant function.
    1. Mask the image pixel according to the specified patches to mask from the custom masker.
    2. Predict the model output for all combinations of masked image and tokens. This is then further passed to the shap li
    """
    with torch.no_grad():
        input_ids = torch.tensor(x[:, :inputs["input_ids"].shape[1]])
        masked_image_token_ids = torch.tensor(x[:, inputs["input_ids"].shape[1]:])
        if use_cuda:
            input_ids = input_ids.cuda()
            masked_image_token_ids = masked_image_token_ids.cuda()
        result = np.zeros(input_ids.shape[0])
        row_cols = 16
        patch_size=14
        for i in range(inputs['input_ids'].shape[0]):
          masked_text_inputs = inputs.copy()
          masked_text_inputs['input_ids'] = input_ids[i].unsqueeze(0)
          masked_image = np.array(image)
          for k in range(masked_image_token_ids[i].shape[0]):
            if masked_image_token_ids[i][k] == 0:  # should be zero
                    m = k // row_cols  # 384 (img shape) / 16 (patch size)
                    n = k % row_cols
                    masked_image[ m * patch_size:(m+1)*patch_size, n*patch_size:(
                        n+1)*patch_size,:] = 0
          imaged = Image.fromarray(masked_image)
          modified_text_clean = processor.tokenizer.decode( masked_text_inputs['input_ids'][0].cuda(), skip_special_tokens=
          messages = [
              {
                 "role": "user",
                 "content": [
                    {"type": "image", "image": imaged},
                    {"type": "text", "text": modified_text_clean}
                    ]}]
          text = processor.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
          inputd = processor(
              text=text,images=imaged,return_tensors="pt",padding=True).to("cuda")
          outputs = model.generate(**inputd, return_dict_in_generate=True, output_scores=True,do_sample=False,       # Gree
               max_new_tokens=1,temperature=0.1)
          yes_tokens = []
          no_tokens = []

          for variant in ["yes", " yes", "Yes", " Yes", "YES", " YES"]:
                token_ids = processor.tokenizer.encode(variant, add_special_tokens=False)
                if token_ids:
                   yes_tokens.append(token_ids[0])

          for variant in ["no", " no", "No", " No", "NO", " NO"]:
                token_ids = processor.tokenizer.encode(variant, add_special_tokens=False)
                if token_ids:
                   no_tokens.append(token_ids[0])
```

```
        yes_tokens = list(set(yes_tokens))
        no_tokens = list(set(no_tokens))
        answer_probs = torch.softmax(outputs.scores[0][0], dim=0)
        yes_prob = sum([answer_probs[tid].item() for tid in yes_tokens])
        no_prob = sum([answer_probs[tid].item() for tid in no_tokens])
        if label==1:
          result[i] = yes_prob
        else:
          result[i] = no_prob
    return result
```

```
!pip install qwen_vl_utils
```

```
Collecting qwen_vl_utils
  Downloading qwen_vl_utils-0.0.14-py3-none-any.whl.metadata (9.0 kB)
Collecting av (from qwen_vl_utils)
  Downloading av-16.0.1-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (4.6 kB)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from qwen_vl_utils) (25.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (from qwen_vl_utils) (11.3.0)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from qwen_vl_utils) (2.32.4)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->qwen_vl_u
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->qwen_vl_utils) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->qwen_vl_utils)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->qwen_vl_utils)
Downloading qwen_vl_utils-0.0.14-py3-none-any.whl (8.1 kB)
Downloading av-16.0.1-cp312-cp312-manylinux_2_28_x86_64.whl (40.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.5/40.5 MB 22.9 MB/s eta 0:00:00
Installing collected packages: av, qwen_vl_utils
Successfully installed av-16.0.1 qwen_vl_utils-0.0.14
```

```
!pip install torchvision
```

```
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-packages (0.24.0+cu126)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torchvision) (2.0.2)
Requirement already satisfied: torch==2.9.0 in /usr/local/lib/python3.12/dist-packages (from torchvision) (2.9.0+cu126)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchvision) (11.3.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torchvision) (3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torc
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torchvision) (75.2.
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torchvision) (1.
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torchvision) (
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torchvision) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torchvision) (20
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch==2.9
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->t
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->t
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->tc
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0-
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0-
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torch
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->tc
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torc
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0-
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->t
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch==2.9.0->torchvision) (3.
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch==2.9
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch==2.9.0->torchv
```

```
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

def colour_coding(shap_values,ids,words):
   fig, ax = plt.subplots(figsize=(10, 6))
   codified = []
   for i in range(len(shap_values[0][0]) - 256):
     if ids[0][i] >= 151644 and ids[0][i] <= 151656:
       continue
     else:
       codified.append(shap_values[0][0][i])
   cmap = matplotlib.colormaps.get_cmap('Reds')
   norm = plt.Normalize(min(codified), max(codified))

   x_position = 0.05
   y_position = 2.5

   for i, word in enumerate(words):
     if (i>=0 and i<len(shap_values[0][0]) - 256) and ( (words[i]=='.' and words[i+1] != '\n') or words[i]=='\n' ):
```

```
      y_position = y_position - 0.1
      x_position = 0.05
      i=i+2
      continue
     color = cmap(norm(codified[i]))

     ax.text(x_position, y_position, word,
            fontsize=30,
            color=color,
            weight='bold',
            transform=ax.transAxes)

     x_position += (len(word) * 0.03 + 0.035)

   ax.set_xlim(0, 1)
   ax.set_ylim(0, 1)
   ax.axis('off')
   plt.show()
```

```
import math
import numpy as np
from qwen_vl_utils import process_vision_info
from IPython.display import display
results = {
        'mmscore': [],
}
shaps=[]
inp=[]
word=[]
for i in range(len(df)):

    test_sentences = [df.iloc[i]['summaries']]
    label = df.iloc[i]['verdict']
    test_img_path = f"/content/image{i+1}.jpg"
    image = Image.open(test_img_path)
    image = image.resize((224, 224), Image.BICUBIC)
    image = image.convert("RGB")


    for k,sentence in enumerate(test_sentences):
          prompt = f"""
You are an impartial criminal law judge deciding a bail application.

Based ONLY on the facts provided , decide whether bail should be granted.

Evidence description:
"{sentence}"
You MUST apply the following principles:
1. Bail is the rule; jail is the exception.
2. Presumption of innocence applies until conviction.
3. Consider BOTH aggravating and mitigating factors.

Aggravating factors may include:
- Strong prima facie evidence
- Risk of absconding
- Threat to witnesses
- Repeat or organized criminal activity
- Seriousness of offence affecting public order

Mitigating factors may include:
- No criminal antecedents
- Judicial custody already undergone
- Weak, general, or uncorroborated allegations
- Offence not punishable with death or life imprisonment
- No risk of tampering or absconding shown

Decision rule:
- Grant bail if mitigating factors outweigh aggravating factors.
- Deny bail only if custody is necessary for justice or public safety.


Respond with ONLY:
"yes" → bail should be granted
"no" → bail should be denied
"""

          conversation = [
                    {
                      "role": "user",
                      "content": [
                          {"type": "image", "image": image},
                          {"type": "text", "text": prompt},
```

```
                ],
            }]
text = processor.apply_chat_template(
      conversation, tokenize=False, add_generation_prompt=True
)
image_inputs, video_inputs = process_vision_info(conversation)
inputs = processor(
      text=[text],
      images=image_inputs,
      videos=video_inputs,
      padding=True,
      return_tensors="pt",
 )
inputs = inputs.to("cuda")

nb_text_tokens = inputs['input_ids'].shape[1]
patch_size = 14

generated_ids = model.generate(**inputs,return_dict_in_generate=True,
                               output_scores=True,
                               do_sample=True,
                               max_new_tokens=2,
                               temperature=0.7)


# Get all possible token IDs for yes/no
yes_tokens = []
no_tokens = []

for variant in ["yes", " yes", "Yes", " Yes", "YES", " YES", "\nyes", "\nYes"]:
     token_ids = processor.tokenizer.encode(variant, add_special_tokens=False)
     if token_ids:
        yes_tokens.append(token_ids[0])

for variant in ["no" ," no", "No", " No", "NO", " NO","\nno", "\nNo"]:
     token_ids = processor.tokenizer.encode(variant, add_special_tokens=False)
     if token_ids:
        no_tokens.append(token_ids[0])

# Remove duplicates
yes_tokens = list(set(yes_tokens))
no_tokens = list(set(no_tokens))


# Now get probabilities for ALL of them
answer_probs = torch.softmax(generated_ids.scores[0][0], dim=0)

yes_prob = sum([answer_probs[tid].item() for tid in yes_tokens])
no_prob = sum([answer_probs[tid].item() for tid in no_tokens])

h,w = 224,224
h_cols = math.ceil(h/14)
w_cols = math.ceil(w/14)
image_token = torch.tensor(range(1,h_cols*w_cols +1)).unsqueeze(0)
X = torch.cat( (inputs["input_ids"].cuda(), image_token.cuda()), 1).unsqueeze(1)
explainer = shap.Explainer(
          get_model_prediction, custom_masker, silent=True,max_evals=4000)
shap_values = explainer(X.cpu())
shaps.append(shap_values.values)
inp.append(inputs['input_ids'])

mm_score = compute_mm_score(nb_text_tokens, shap_values)
results["mmscore"].append(mm_score)
print(mm_score)
words = [] # Initialize decoded_tokens to ensure it's fresh for each run
for token_id_tensor in inputs['input_ids'][0]: # Iterate through each token ID tensor in the first sequence
     token_id = token_id_tensor.item() # Get the Python scalar value from the tensor
     if token_id < 151643 or token_id > 151656:
          asd = processor.tokenizer.decode([token_id], skip_special_tokens=False)
          words.append(asd)
word.append(words)
```

```
0.6965887924645087
0.7875948982039485
0.7449664349636871
0.8039855067320959
0.7101937516934356
```

```
for i in range(len(shaps)):
  colour_coding(shaps[i],inp[i],word[i])
```