1st

Model Explore

```
!pip install -q transformers accelerate bitsandbytes huggingface_hub
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.9/72.9 MB 11.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 69.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 52.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 39.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 17.0 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 8.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 73.5 MB/s eta
0:00:00
```

```
from huggingface_hub import notebook_login
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch



notebook_login()
```


```
{"model_id":"10425b3c448a4df8ac24d8d517972711","version_major":2,"vers
ion_minor":0}
```


more model

```
torch.cuda.is_available()
```

```
True
```

```
model_id = "meta-llama/Llama-2-7b-hf"
```

```
# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

```
/usr/local/lib/python3.11/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 it as
secret in your Google Colab and restart your session.
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(
```


```
{"model_id":"97f5187413744b72be942d12416053e0","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"5c778ff05cc744f297e5a6c2b6410b53","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"694969fa467a4318b0fa93e4e8e772e3","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"40c1c22a3bb94c76833eb640761f0c20","version_major":2,"vers
ion_minor":0}
```


```
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
```

```
# Load model with 8-bit precision (needs less memory)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map="auto",
    load_in_8bit=True,  # For lower RAM usage (needs `bitsandbytes`)
    torch_dtype=torch.float16,
)
```


```
{"model_id":"e0560192052647fcb075b2d7e1e1b0c3","version_major":2,"vers
ion_minor":0}
```


```
The `load_in_4bit` and `load_in_8bit` arguments are deprecated and
will be removed in the future versions. Please, pass a
`BitsAndBytesConfig` object in `quantization_config` argument instead.
```


```
{"model_id":"3f768a7499f848a8ad03029b8226e0bd","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"1a572994501a4a3186e5f2a5e30cf002","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"fd71515171f74848b42e6f3df108457a","version_major":2,"vers
ion_minor":0}
```

{"model_id":"1af866845d1c4bbe9d0917e1392681c6","version_major":2,"version_minor":0}

{"model_id":"3e3f0e3b496f4af2b6d745a3a5799467","version_major":2,"version_minor":0}

{"model_id":"a527c0fc65514754bf7a849804ff26a6","version_major":2,"version_minor":0}

```
def answer(prompt):
    inputs = tokenizer(prompt, return_tensors="pt").to("cuda")
    # output = model.generate(**inputs, max_new_tokens=200)
    outputs = model.generate(
        **inputs,
        max_new_tokens=150,
        do_sample=True,
        temperature=0.7,
        top_p=0.9,
        top_k=50,
        eos_token_id=tokenizer.eos_token_id)
    # print(tokenizer.decode(outputs[0], skip_special_tokens=True))
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

prompt = " I have knee pain which medicine should i take?"
answer(prompt)
```

{"type":"string"}

```
prompt = "I have headache which medicine should i take?"
answer(prompt)
```

{"type":"string"}

detection

```
!pip install -q wikipedia
```

```
  Preparing metadata (setup.py) ...
```

```
from transformers import  pipeline
import torch
import nltk
import wikipedia

nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
True
```

```
# Setup NLI verification pipeline

nli = pipeline(
    "text-classification",
    model="ynie/roberta-large-snli_mnli_fever_anli_R1_R2_R3-nli"
)

nltk.download('punkt')  # for sentence tokenization


#  Wikipedia context retrieval

def get_wikipedia_context(query: str, sentences: int = 5) -> str:
    try:
        return wikipedia.summary(query, sentences=sentences)
    except Exception as e:
        print(f"[Warning] Could not fetch Wikipedia summary for
'{query}': {e}")
        return ""
```

{"model_id":"00fce8572ad64f9e842a1c368f81e483","version_major":2,"version_minor":0}

{"model_id":"dd8245bc79f045a0a554743501cae5f5","version_major":2,"version_minor":0}

{"model_id":"48af4263682846bbb80de474e5d58323","version_major":2,"version_minor":0}

```
Some weights of the model checkpoint at ynie/roberta-large-
snli_mnli_fever_anli_R1_R2_R3-nli were not used when initializing
RobertaForSequenceClassification: ['roberta.pooler.dense.bias',
'roberta.pooler.dense.weight']
- This IS expected if you are initializing
RobertaForSequenceClassification from the checkpoint of a model
trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing
RobertaForSequenceClassification from the checkpoint of a model that
you expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
```

{"model_id":"d0926cc219ca43818277a40ce84271f7","version_major":2,"version_minor":0}

{"model_id":"8434575819894f489230650eb1edd50e","version_major":2,"version_minor":0}

{"model_id":"19e3c0e4b12b418fa3954408ae56a1ba","version_major":2,"version_minor":0}

{"model_id":"3879227b8e014af1beceebf133d8dd1c","version_major":2,"version_minor":0}

```
Device set to use cuda:0
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```python
# Hallucination detection function

def detect_hallucinations(answer: str, topic: str) -> list:
    # Retrieve premise
    context = get_wikipedia_context(topic)

    print("-----------")
    print(context)
    print("-----------")

    # if not context:
    #     print("No Wikipedia context available. Skipping verification.")
    #     return []

    # Tokenize into sentences
    print("-----------")
    sentences = nltk.sent_tokenize(answer)
    print(sentences)
    print("-----------")

    results = []

    # NLI check each
    for sent in sentences:
        nli_input = context + " [SEP] " + sent
        res = nli(nli_input)[0]
        results.append({
            "sentence": sent,
            "label": res["label"],
            "score": res["score"]
        })
    return results


# Step 5: Example end-to-end

if __name__ == "__main__":
```

```
    topic = "knee pain medication"
    # Generate an answer using llama from above
    prompt = f"Provide advice on {topic}."
    generated_answer = answer(prompt)
    print("\n[Generated Answer]\n", generated_answer)




    # Detect hallucinations
    detections = detect_hallucinations(generated_answer, topic)
    print("\n[Hallucination Checks]")
    for item in detections:
        print(f"Sentence: {item['sentence']}")
        print(f"  Label: {item['label']} (confidence:
{item['score']:.2f})\n")
```

```
[Generated Answer]
 Provide advice on knee pain medication.
In some cases, knee pain medication is prescribed to relieve the
symptoms of knee pain.
The knee pain medication prescribed by your doctor will depend on the
cause of your knee pain and the severity of your symptoms.
You may be prescribed knee pain medication if your knee pain is severe
or if you have other medical conditions that require pain relief.
Knee pain medication can help reduce inflammation and swelling in the
joints, which can relieve some of the pain associated with knee pain.
It can also help to relieve the pain associated with other conditions
that cause knee pain, such as arthritis or tendinitis
-----------
Osteoarthritis is a type of degenerative joint disease that results
from breakdown of joint cartilage and underlying bone. A form of
arthritis, it is believed to be the fourth leading cause of disability
in the world, affecting 1 in 7 adults in the United States alone. The
most common symptoms are joint pain and stiffness. Usually the
symptoms progress slowly over years. Other symptoms may include joint
swelling, decreased range of motion, and, when the back is affected,
weakness or numbness of the arms and legs.
-----------
-----------
['Provide advice on knee pain medication.', 'In some cases, knee pain
medication is prescribed to relieve the symptoms of knee pain.', 'The
knee pain medication prescribed by your doctor will depend on the
cause of your knee pain and the severity of your symptoms.', 'You may
be prescribed knee pain medication if your knee pain is severe or if
you have other medical conditions that require pain relief.', 'Knee
pain medication can help reduce inflammation and swelling in the
```

```
joints, which can relieve some of the pain associated with knee
pain.', 'It can also help to relieve the pain associated with other
conditions that cause knee pain, such as arthritis or tendinitis']
-----------

/usr/local/lib/python3.11/dist-packages/torch/nn/modules/
module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated
and will be removed in version 4.55.0 for
`RobertaSdpaSelfAttention.forward`.
  return forward_call(*args, **kwargs)


[Hallucination Checks]
Sentence: Provide advice on knee pain medication.
  Label: neutral (confidence: 1.00)

Sentence: In some cases, knee pain medication is prescribed to relieve
the symptoms of knee pain.
  Label: neutral (confidence: 1.00)

Sentence: The knee pain medication prescribed by your doctor will
depend on the cause of your knee pain and the severity of your
symptoms.
  Label: neutral (confidence: 0.95)

Sentence: You may be prescribed knee pain medication if your knee pain
is severe or if you have other medical conditions that require pain
relief.
  Label: neutral (confidence: 0.98)

Sentence: Knee pain medication can help reduce inflammation and
swelling in the joints, which can relieve some of the pain associated
with knee pain.
  Label: neutral (confidence: 1.00)

Sentence: It can also help to relieve the pain associated with other
conditions that cause knee pain, such as arthritis or tendinitis
  Label: neutral (confidence: 1.00)
```

start

```
import torch
import torch.nn.functional as F
import re

# 3) Generation + confidence collector
def answer_with_confidence(prompt, max_new_tokens=150,
                           temperature=0.7, top_p=0.9, top_k=50):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    input_len = inputs["input_ids"].size(-1)

    out = model.generate(
        **inputs,
        max_new_tokens=max_new_tokens,
        do_sample=True,
        temperature=temperature,
        top_p=top_p,
        top_k=top_k,
        eos_token_id=tokenizer.eos_token_id,
        output_scores=True,
        return_dict_in_generate=True,
    )
    seq = out.sequences[0]
    gen_ids = seq[input_len:].tolist()

    # compute token log-probs
    token_logps = []
    for logits, tid in zip(out.scores, gen_ids):
        lp = F.log_softmax(logits, dim=-1)[tid].item()
        token_logps.append(lp)

    text = tokenizer.decode(seq, skip_special_tokens=True)
    return text, token_logps

# 4) Sentence-level detector
def detect_low_confidence_sentences(text, token_logps, threshold=-
4.5):
    # split into sentences (keeping punctuation)
    parts = re.split(r'([.?!])', text)
    sentences = ["".join(parts[i:i+2]).strip()
                 for i in range(0, len(parts)-1, 2)]

    flagged = []
    idx = 0
    for sent in sentences:
        toks = tokenizer(sent, return_tensors="pt")["input_ids"]
[0].tolist()
        lps = token_logps[idx: idx + len(toks)]
        idx += len(toks)
```

```
        if not lps:
            continue
        avg_lp = sum(lps) / len(lps)
        if avg_lp < threshold:
            flagged.append((sent, avg_lp))
    return flagged

# 5) Build a 20-question medical prompt set
medical_prompts = [
    "What is the first-line treatment for hypertension?",
    "How is type 2 diabetes diagnosed?",
    "What are common side effects of amoxicillin?",
    "Which vaccine protects against HPV?",
    "How do you manage acute asthma exacerbation?",
    "What is the mechanism of action of beta-blockers?",
    "Describe the presentation of myocardial infarction.",
    "What lab test confirms hypothyroidism?",
    "How is osteoporosis prevented in postmenopausal women?",
    "What is the recommended dose of aspirin for cardioprotection?",
    "How does insulin regulate blood sugar?",
    "What are the signs of stroke?",
    "Which antibiotic is used for MRSA skin infections?",
    "How do you treat acute otitis media in children?",
    "What are the risk factors for deep vein thrombosis?",
    "What is the gold standard for diagnosing pulmonary embolism?",
    "How is chronic kidney disease staged?",
    "What lifestyle changes help in managing hyperlipidemia?",
    "Which drug class is used for GERD?",
    "How is iron-deficiency anemia treated?"
]

# 6) Run the detector on each
threshold = -4.5
for prompt in medical_prompts:
    print(f"\n=== Prompt: {prompt}")
    text, logps = answer_with_confidence(prompt)
    flagged = detect_low_confidence_sentences(text, logps, threshold)
    if not flagged:
        print("  □ No low-confidence sentences detected.")
    else:
        print("  ⚠ Flagged sentences:")
        for sent, lp in flagged:
            print(f"    • \"{sent}\"  (avg log-prob {lp:.2f})")


=== Prompt: What is the first-line treatment for hypertension?

---------------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
```

```
last)
/tmp/ipython-input-37-1630500704.py in <cell line: 0>()
      3 for prompt in medical_prompts:
      4     print(f"\n=== Prompt: {prompt}")
----> 5     text, logps = answer_with_confidence(prompt)
      6     flagged = detect_low_confidence_sentences(text, logps,
threshold)
      7     if not flagged:

/tmp/ipython-input-35-2113541463.py in answer_with_confidence(prompt,
max_new_tokens, temperature, top_p, top_k)
     22     token_logps = []
     23     for logits, tid in zip(out.scores, gen_ids):
---> 24         lp = F.log_softmax(logits, dim=-1)[tid].item()
     25         token_logps.append(lp)
     26

IndexError: index 13 is out of bounds for dimension 0 with size 1
```

```python
# Corrected answer_with_confidence with proper squeezing of logits
before indexing

import torch.nn.functional as F

def answer_with_confidence(prompt, max_new_tokens=150,
                           temperature=0.7, top_p=0.9, top_k=50):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    input_len = inputs["input_ids"].size(-1)

    out = model.generate(
        **inputs,
        max_new_tokens=max_new_tokens,
        do_sample=True,
        temperature=temperature,
        top_p=top_p,
        top_k=top_k,
        eos_token_id=tokenizer.eos_token_id,
        output_scores=True,
        return_dict_in_generate=True,
    )
    seq = out.sequences[0]
    gen_ids = seq[input_len:].tolist()

    # compute token log-probs, correctly handling batched logits
    token_logps = []
    for logits, tid in zip(out.scores, gen_ids):
        # logits shape is [batch_size, vocab_size]; we index batch dim
first
        if logits.dim() == 2:
            logits = logits.squeeze(0)  # now shape [vocab_size]
```

```
        log_probs = F.log_softmax(logits, dim=-1)
        token_logps.append(log_probs[tid].item())

    text = tokenizer.decode(seq, skip_special_tokens=True)
    return text, token_logps

# Now re-run one prompt to verify it works without IndexError:
prompt = "What is the first-line treatment for hypertension?"
text, logps = answer_with_confidence(prompt)
print(text)
```

```
What is the first-line treatment for hypertension?
What medications are used to treat hypertension?
How can I lower my blood pressure quickly?
What are the side effects of high blood pressure medication?
How much water should I drink to lower blood pressure?
How much water should I drink to lower my blood pressure?
What is the best time to drink water to lower blood pressure?
How much water should I drink for high blood pressure?
What is the best time to drink water to lower blood pressure?
What is the best time to drink water for high blood pressure?
How can I lower my blood pressure immediately?
What are the best drinks to lower blood pressure?
What is the best time to drink water for high blood pressure?
```

benchmarking, dataset , hyperparametertuning, with metrices blue score and others fine tune,

```
!pip install -q wikipedia
```