INF8460 – Traitement automatique de la langue naturelle - Automne 2025

TP4: Modèle RAG (Retrieval Augmented Generation) pour la réponse à des questions temporelles (Temporal Question Answering) (100 points)

Identification de l'équipe

Identification de l'équipe:

Groupe de laboratoire: Groupe 3

Equipe numéro : Equipe 10

Membres:

- Aziz Hidri, matricule : 2171081 (33 % de contribution, nature de la contribution)
- Amaury Miquel, matricule : 2487172 (33 % de contribution, nature de la contribution)
- Bazoune Mahdi Fares, matricule : 2236937 (33% de contribution, nature de la contribution)

Nature de la contribution:

Au sein de ce TP, chaque membre du groupe a effectué le TP de son côté puis nous avons réuni les meilleurs méthodes afin de former un colab optimal.

Usage des LLMs

Notez que l'usage des LLMs est strictement prohibé.

Vous devez masquer les fonctionnalités d'IA générative dans les paramètres de Colab (Paramètres -> Assistance IA).

Même si cela demande plus d'effort et de temps, la résolution des TPs par vos propres moyens vous permettra d'acquérir toutes les compétences en TALN pour vos travaux futurs. Nous incluerons des questions de code des TPs dans l'examen final et considérerons toute utilisation de LLM comme un cas de plagiat.

Description du TP

Ce travail porte sur le questionnement temporel en utilisant la génération augmentée par récupération (RAG). Vous devrez initialement vous familiariser avec les données, puis tester différentes méthodes de génération. Finalement, la dernière partie du TP est une compétition dans laquelle vous devez explorer les méthodes de l'état de l'art pour cette tâche, proposer votre propre solution, l'implémenter et la comparer au modèle de référence.

La tâche est la suivante : À partir d'une question en langue naturelle, le modèle doit générer une réponse. Voici un exemple d'entrée et de sortie :

- Input : what was the wedding date of nikolay gumilev and anna akhmatova ?
- Output : [1910]

Les données utilisées proviennent d'un dataset public qui regroupe des questions étiquetées selon leur type temporel (`Explicit` , `Temp.Ans` , `Ordinal` , `Implicit`), avec des réponses liées à Wikidata / Wikipedia. Le jeu de données est divisé en ensembles d'entraînement, de validation et de test. Les différents éléments présents dans les données sont:

- `id` : Identifiant unique pour chaque ligne du dataframe.
- `entities` : Liste d'entités (identifiants de QID de Wikidata) associées à la question.
- `natural_lang_entities` : Liste des entités associées à la question en langage naturel (par exemple, noms de personnes, lieux, etc.).
- `times` : Liste d'instants temporels (dates, années, etc.) associés à la question.
- `question` : La question formulée, généralement liée à un fait ou à une entité.
- `original_answers` : Réponses sous forme d'identifiants QID de Wikidata.
- `natural_lang_answers` : Réponses sous forme de texte en langage naturel.
- `answer_type` : Type de réponse ("entity" pour une entité spécifique, "time" pour une donnée temporelle).
- `type` : Catégorie de la question ("Temp.Ans" pour une réponse temporelle, "Explicit" pour une réponse directe à la question, "Ordinal" pour une réponse nécéssitant d'ordonner des éléments, "Implicit" pour une réponse nécéssitant un raisonnement sur la question).
- `nbhood_facts` : Identifiant des faits et relations concernant directement la question ou les entités de la question.
- `natural_lang_nbhood_facts` : Liste des faits ou relations nbhood exprimés en langage naturel.

Vous disposez également d'une liste d'environ 240 597 faits qui pourraient être utilisés por aider le modèle génératif à répondre aux questions. En voici quelques exemples :

```
'subject': 'Anna Akhmatova', 'relation': 'place of marriage', 'object': 'Kiev', 'start_tim
e': '1910', 'end_time': '1918'
'subject': 'Nikolay Gumilev', 'relation': 'spouse', 'object': 'Anna Akhmatova', 'start_tim
e': '1910', 'end_time': '1918'
'subject': 'Anna Akhmatova', 'relation': 'spouse', 'object': 'Nikolay Punin', 'start_tim
e': '1922', 'end_time': '1935'
'subject': 'Anna Engelhardt', 'relation': 'spouse', 'object': 'Nikolay Gumilev', 'start_ti
me': '1919'
```

Pour résoudre cette tâche, nous utiliserons le modèle Llama-3.2-3B-Instruct (https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct). Pour utiliser ce modèle, il vous faudra:

- Créer un compte HuggingFace (https://huggingface.co/join (https://huggingface.co/join))
- Accepter la licence `COMMUNITY LICENSE AGREEMENT` de Meta pour l'utilisation du modèle (un moment d'attente de quelques minutes peut être nécessaire avant de pouvoir accéder au modèle)
- Créer un Acces Token HuggingFace (le token sera requis à la section 3.1)

Le plan du TP est le suivant :

- Partie 0 : Mise en place
- Partie 1 : Familiarisation avec le jeux de données
- Partie 2 : Module d'évaluation
- Partie 3 : Génération avec mémoire paramétrique
- Partie 4 : RAG
- Partie 5 : Affinage
- Partie 6 : Compétition Kaggle

Partie 0 - Mise en place (0 pts)

Cette section contient le code nécessaire à l'installation des librairies, leur importation, et la définition des jeux de données. Pour les parties 0 à 4, veillez à ne pas utiliser d'autres librairies que celles importées à cette section.

In [ ]:
```
#@title Installation des librairies
!pip install -q accelerate
!pip install -q datasets
!pip install -q bitsandbytes
!pip install -q llama-index
!pip install -q llama-index-embeddings-huggingface
!pip install -q llama-index-llms-huggingface
!pip install -q torch
!pip install -q transformers
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.4/59.4 MB 13.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.9/11.9 MB 140.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 303.3/303.3 kB 22.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.8/51.8 kB 5.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 92.0/92.0 kB 9.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.9/63.9 kB 6.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 329.5/329.5 kB 30.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 80.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.0/88.0 kB 8.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.9/50.9 kB 4.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 150.7/150.7 kB 14.3 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packag
es that are installed. This behaviour is the source of the following dependency conf
licts.
ipython 7.34.0 requires jedi>=0.16, which is not installed.
```

In [ ]:
```
#@title Importation des librairies
import re
import os
import json
import nltk
import time

import torch

import pickle
import random

import numpy as np
import pandas as pd

from tqdm import tqdm

from getpass import getpass
from datasets import Dataset
import matplotlib.pyplot as plt
from huggingface_hub import login
from dataclasses import dataclass
from sklearn.metrics.pairwise import cosine_similarity
from typing import List, Tuple, Dict, Any, Optional, Union
from concurrent.futures import ThreadPoolExecutor, as_completed
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig, Tra
iningArguments, Trainer, DataCollatorForLanguageModeling, pipeline
from peft import LoraConfig, get_peft_model, TaskType, PeftModel, prepare_model_for_k
bit_training
```

In [ ]:
```
#@title Liaison avec Google Drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

La cellule suivant importe les données depuis votre Google Drive. Assurez vous d'ajouter le chemin d'accès au dossier dans votre configuration à la liste `candidates` .

In [ ]:
```python
#@title Importation des données
def get_data_path():
    """
    Detects which Google Drive folder structure is available
    and returns the correct PATH for TP4 data.
    This function is made for colab usage.
    Please add you own path in candidates.
    """
    candidates = [
        "/content/drive/MyDrive/TP4",
        "/content/drive/MyDrive/NLP/TP4_Amaury",
        "/content/drive/MyDrive/INF8460-A25/TP4"
    ]

    for path in candidates:
        if os.path.exists(path):
            return path

    raise FileNotFoundError("None of the candidate data paths exist. Please check you
r Drive mount.")

# Usage
PATH = get_data_path()
DATA_PATH = f"{PATH}/data"
# Make sure the cache folder exists
os.makedirs(f"{DATA_PATH}/cache", exist_ok=True)
print("Using PATH:", PATH)


SPLITS = ['train', 'test', 'val']

dataset = {
      split: pd.read_csv(f"{DATA_PATH}/{split}.csv")
      for split in SPLITS
}

# Set facts_dataframe
with open(f"{DATA_PATH}/facts.json", "r", encoding="utf-8") as f:
    list_of_facts = [json.loads(line) for line in f][0]
facts_dataframe = pd.DataFrame({
    'id': [id for id in range(len(list_of_facts))],
    'fact': [str(fact) for fact in list_of_facts],
})

# Set label_mapping_dataframes
with open(f"{DATA_PATH}/label_mappings.json") as f:
    data = json.load(f)
label_mapping_dataframes = pd.DataFrame(list(data.items()), columns=["id", "label"])
```

```
Using PATH: /content/drive/MyDrive/NLP/TP4_Amaury
```

Visualisation rapide

La cellule suivante affiche les 5 premières lignes du jeu d'entraînement.

In [ ]:
```
dataset['train'].head(n=5)
```

Out[ ]:

	id	entities	natural_lang_entities	times	question	original_answers	natural_lang_answers	answer_
0	1	['Q267400', 'Q824192']	['Rachael Harris', 'The Daily Show']	[]	what is the end time for the daily show as rac...	[2003]	['2003']	
1	2	['Q11033', 'Q17452']	['mass media', 'Grand Theft Auto V']	[]	what medium is preceded by grand theft auto v	['Q5294']	['DVD']	
2	3	['Q291169']	['Jeremy Corbyn']	[]	what is jeremy corbyn's position and when was ...	['Q918472', 'Q543609', 'Q918466', 'Q25052149',...	['1987 United Kingdom general election', '1992...	
3	5	['Q311033', 'Q80440']	['Nikolay Gumilev', 'Anna Akhmatova']	[]	what was the wedding date of nikolay gumilev a...	[1910]	['1910']	
4	7	['Q3306168', 'Q1434880', 'Q30']	['Charles Everett Koop', 'Surgeon General of t...	[]	until when was c. everett koop the surgeon gen...	[1989]	['1989']	

0.1 Fonctions outils (0 point)

On met à votre disposition les deux fonctions `get_factID_from_factLabel` et `get_factLabel_from_factID` suivantes qui permettent de récupérer l'identifiant d'un document à partir de son label et vice-versa.

In [ ]:
```
def get_factID_from_factLabel(label, facts_dataframe=facts_dataframe):
  # Ensure the input label is a string for comparison
  label_str = str(label)
  # Filter the dataframe and get the 'id'
  result = facts_dataframe[facts_dataframe['fact']==label_str]['id'].to_list()
  if not result:
      # Handle cases where the fact label is not found
      # print(f"Warning: Fact label '{label_str}' not found in provided facts_datafra
me.")
      return None # Or raise an error, depending on desired behavior
  return result[0]

def get_factLabel_from_factID(factID, facts_dataframe=facts_dataframe):
  return facts_dataframe[facts_dataframe['id']==factID]['fact'].to_list()[0]
```

Partie 1 - Statistiques sur les données (10 pts)

Après avoir téléchargé les données, vous devez maintenant vous familiariser avec leur contenu.

Q1.1 (1 pts) - Affichez un échantillon aléatoire de chaque ensemble (train, test, val)

In [ ]:

```
### TODO ###
for split in SPLITS:
    print(f"\n=== Échantillon aléatoire ({split}) ===")
    display(dataset[split].sample(n=5, random_state=42))
### END TODO ###
```

=== Échantillon aléatoire (train) ===

	id	entities	natural_lang_entities	times	question	original_answers	natural_lang_ar
132	185	['Q182486', 'Q8445', 'Q18623']	['Greer Garson', 'marriage', '1943']	[1943]	what is spouse of greer garson that is start t...	['Q361523']	['Richar
3235	4507	['Q15338', 'Q1639492']	['East Godavari district', 'Rajahmundry']	[]	when did rajamahendravaram was located as an a...	[1925]	
2168	3043	['Q555271', 'Q1035067']	['Reggie Bush', 'Heisman Trophy']	[]	what year did reggie bush win the heisman trophy	[2005]	
4085	5696	['Q103949', 'Q561852']	['Buster Keaton', 'Willie Mays']	[1921]	on may 31, 1921, who wed buster keaton	['Q444788']	['Natalie Taln
4409	6149	['Q1156895', 'Q1101377']	['nomination', 'Clifton Chenier']	[1983]	in 1983, what nomination did clifton chenier r...	['Q5593797']	['Grammy Aw Best Et Traditi

=== Échantillon aléatoire (test) ===

	id	entities	natural_lang_entities	times	question	answer_type	type
844	13789	['Q265538', 'Q928040']	['World Series', '2010 World Series']	[2003]	what year did the champion of the 2003 world s...	entity	['Temp.Ans']
2407	15352	['Q131371']	['Boston Celtics']	[]	which player on the boston celtics' current ro...	entity	['Explicit']
1427	14372	['Q4295898', 'Q650816']	['1976 World Series', 'Baltimore Orioles']	[100]	what year did the orioles win 100 games	entity	['Temp.Ans']
321	13266	['Q12003', 'Q8445']	['Cher', 'marriage']	[1979]	what is spouse of cher that is end time is 197...	entity	['Explicit']
2839	15784	['Q265538']	['World Series']	[]	in what years has the team that plays in ed sm...	entity	['Temp.Ans']

=== Échantillon aléatoire (val) ===

	id	entities	natural_lang_entities	times	question	original_answers	natural_lang_answers	a
844	10553	['Q182882']	['John James Audubon']	[1827]	what is award received of john james audubon t...	['Q5438598']	['Fellow of the Royal Society of Edinburgh']	
2406	12115	['Q32096', 'Q204862']	['Super Bowl', 'Dallas Cowboys']	[]	what years have the dallas cowboys won the sup...	['Q621500', 'Q1065229', 'Q632774', 'Q1064741',...	['Super Bowl XXVIII', 'Super Bowl XXX', 'Super...	
1427	11136	['Q30', 'Q816']	['United States of America', 'Arizona']	[2009]	which arizona governor holds a position in the...	['Q236941', 'Q229032']	['Jan Brewer', 'Janet Napolitano']	
321	10030	['Q32096']	['Super Bowl']	[]	when is the last time the the sports team at p...	['Q853768']	['Super Bowl XLIII']	
2838	12547	['Q38104', 'Q36488']	['Nobel Prize in Physics', 'Guglielmo Marconi']	[]	on what date did guglielmo marconi receive the...	[1909, '+69900']	['1909', '+69900']	

Q1.2 (1 pts) - Combien de questions y a-t-il dans chaque ensemble (train, validation, test) ?

In [ ]:
```
### TODO ###
for split in SPLITS:
    print(f"{split} : {len(dataset[split])} questions")
### END TODO ###
```

```
train : 6970 questions
test : 3237 questions
val : 3236 questions
```

Q1.3 (1 pts) Quel est le nombre total de faits uniques dans l'ensemble des données de faits ?

In [ ]:
```
### TODO ###
nb_faits_uniques = facts_dataframe['fact'].nunique()
print("Nombre total de faits uniques :", nb_faits_uniques)
### END TODO ###
```

```
Nombre total de faits uniques : 239324
```

Q1.4 (1 pts) Combien d'entités uniques sont présentes dans les données d'entités ?

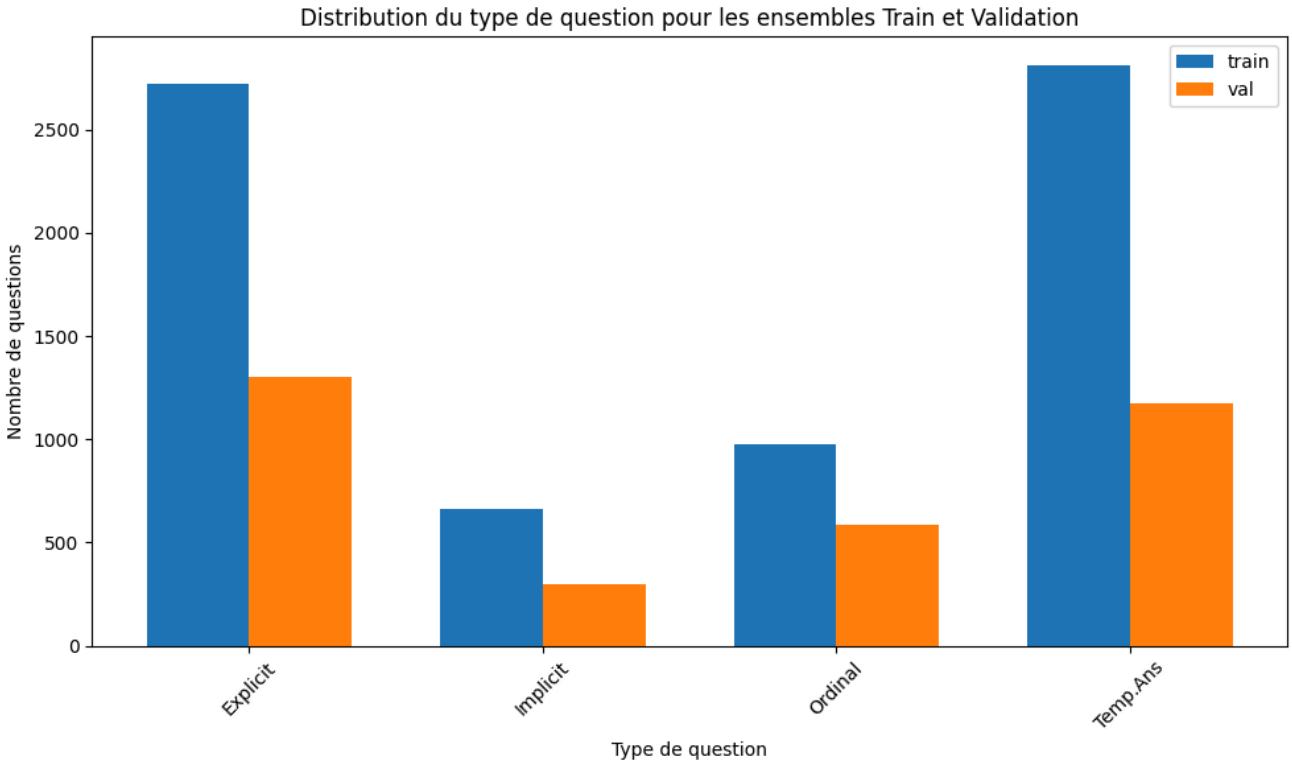
In [ ]:
```python
unique_entities = set()
for split in SPLITS:
    for entity_list in dataset[split]['entities']:
        if entity_list is None:
            continue
        if isinstance(entity_list, list):
            unique_entities.update(entity_list)
        elif isinstance(entity_list, str):
            unique_entities.add(entity_list)

print("Nombre d'entités uniques présentes dans l'ensemble des données :")
print(len(unique_entities))
```

```
Nombre d'entités uniques présentes dans l'ensemble des données :
7903
```

Q1.5 (2 pts) Tracez un graphe montrant la distribution de type de question (`type`) pour les splits de train et val (un seul graphe).

In [ ]:

```
### TODO ###
import ast

def extract_types(df):
    all_types = []
    for val in df['type']:
        try:
            parsed = ast.literal_eval(val)
            if isinstance(parsed, list):
                all_types.extend(parsed)
            else:
                all_types.append(parsed)
        except Exception:
            all_types.append(val)
    return all_types

# Count occurrences in train and val
train_counts = pd.Series(extract_types(dataset['train'])).value_counts()
val_counts   = pd.Series(extract_types(dataset['val'])).value_counts()

# Align both series (to avoid missing categories)
all_types = sorted(set(train_counts.index).union(val_counts.index))
train_vals = [train_counts.get(t, 0) for t in all_types]
val_vals   = [val_counts.get(t, 0) for t in all_types]

# Plot
x = np.arange(len(all_types))
width = 0.35

plt.figure(figsize=(10,6))
plt.bar(x - width/2, train_vals, width, label='train')
plt.bar(x + width/2, val_vals, width, label='val')

plt.title("Distribution du type de question pour les ensembles Train et Validation")
plt.xlabel("Type de question")
plt.ylabel("Nombre de questions")
plt.xticks(x, all_types, rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
### END TODO ###
```



Q1.6 (1 pts) - Étudiez la distribution (min, max, moyenne, écart-type) du nombre d'entités associées et du nombre d'éléments dans les réponses (nombre de réponse par question), par question pour le jeu d'entraînement.

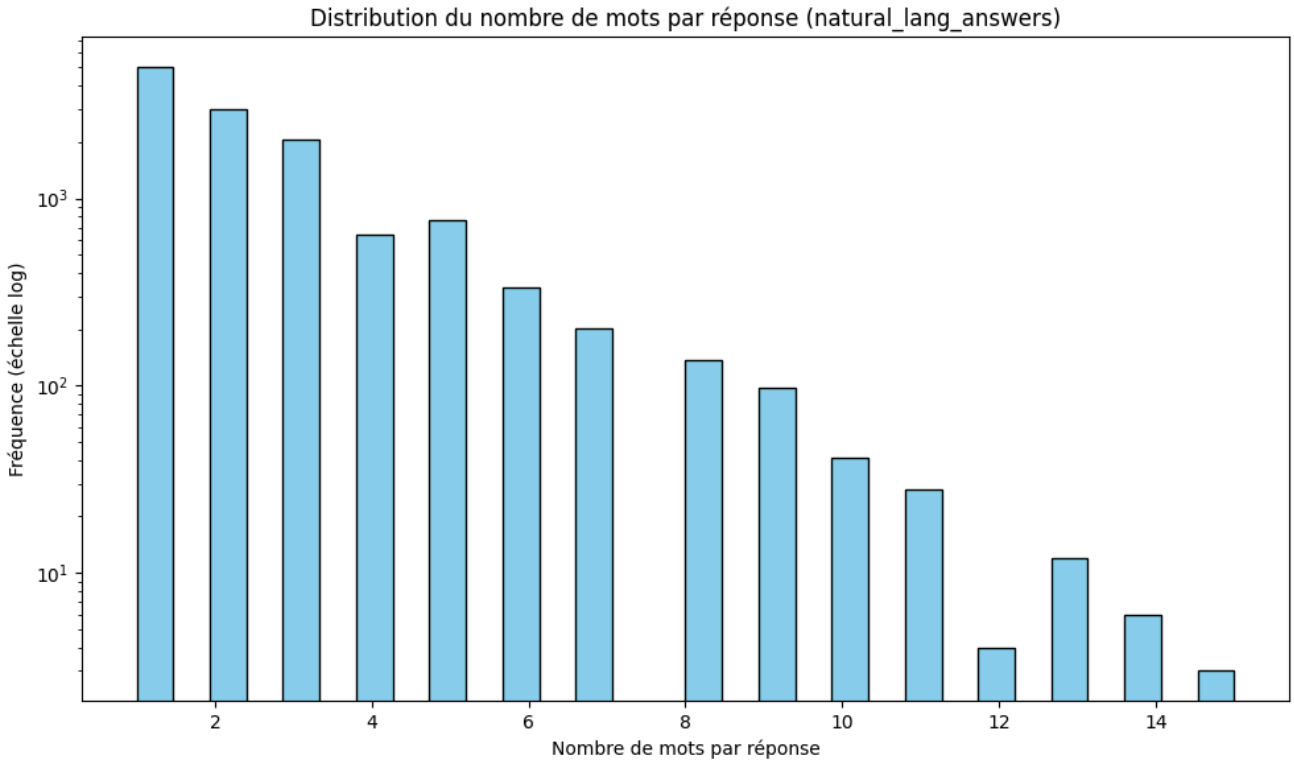
In [ ]:
```python
### TODO ###
import ast

def count_list_elements(series):
    counts = []
    for val in series:
        try:
            parsed = ast.literal_eval(val)
            if isinstance(parsed, list):
                counts.append(len(parsed))
            else:
                counts.append(1)
        except Exception:
            # Si ce n'est pas une liste valide, on considère 0
            counts.append(0)
    return np.array(counts)

# Nombre d'entités par question (colonne 'entities')
entity_counts = count_list_elements(dataset['train']['entities'])

# Nombre de réponses par question (colonne 'original_answers')
answer_counts = count_list_elements(dataset['train']['original_answers'])

min_entities, max_entities, mean_entities, std_entities = (
    np.min(entity_counts), np.max(entity_counts),
    np.mean(entity_counts), np.std(entity_counts)
)

min_answers, max_answers, mean_answers, std_answers = (
    np.min(answer_counts), np.max(answer_counts),
    np.mean(answer_counts), np.std(answer_counts)
)
### END TODO ###

print(f"\nStatistiques pour le jeu d'entraînement:")
print(f"Nombre d'entités par question (min, max, moyenne, écart-type): ({min_entities}, {max_entities}, {mean_entities:.2f}, {std_entities:.2f})")
print(f"Nombre de réponses par question (min, max, moyenne, écart-type): ({min_answers}, {max_answers}, {mean_answers:.2f}, {std_answers:.2f})")
```

```
Statistiques pour le jeu d'entraînement:
Nombre d'entités par question (min, max, moyenne, écart-type): (0, 6, 1.87, 0.77)
Nombre de réponses par question (min, max, moyenne, écart-type): (1, 233, 1.78, 3.9
1)
```

Q1.7 (1 pts) - Montrez (à l'aide d'un graphique) la distribution du nombre de mots par réponse en langue naturelle (`natural_lang_answers`) dans le jeu de données d'entraînement.

In [ ]:

```python
### TODO ###
def extract_all_answers(series):
    answers = []
    for val in series:
        try:
            parsed = ast.literal_eval(val)
            if isinstance(parsed, list):
                answers.extend([str(a) for a in parsed])
            else:
                answers.append(str(parsed))
        except Exception:
            answers.append(str(val))
    return answers

# Extraire toutes les réponses textuelles
all_answers = extract_all_answers(dataset['train']['natural_lang_answers'])

# Compter le nombre de mots par réponse
word_counts = [len(ans.split()) for ans in all_answers if isinstance(ans, str)]

# Tracer la distribution
plt.figure(figsize=(10,6))
plt.hist(word_counts, bins=30, color='skyblue', edgecolor='black')
plt.yscale('log')  # optionnel : met la fréquence en échelle log
plt.title("Distribution du nombre de mots par réponse (natural_lang_answers)")
plt.xlabel("Nombre de mots par réponse")
plt.ylabel("Fréquence (échelle log)")
plt.tight_layout()
plt.show()

### END TODO ###
```



Q1.8 (2 pts) - En considérant la colonne `question`, vérifiez l'intersection entre (les éléments des jeux d'entraînement combinés avec ceux de validation) et ceux du jeu de test afin de savoir s'il y a une fuite de données.

In [ ]:
```python
### TODO ###
train_questions = set(dataset['train']['question'])
val_questions   = set(dataset['val']['question'])
test_questions  = set(dataset['test']['question'])

# On combine train et val
train_val_questions = train_questions.union(val_questions)

# Intersection avec test
intersection = train_val_questions.intersection(test_questions)

# Résultats
nb_common = len(intersection)
total_test = len(test_questions)
pourcentage = (nb_common / total_test) * 100

print(f"Nombre de questions communes entre (train+val) et test : {nb_common}")
print(f"Pourcentage de fuite potentielle : {pourcentage:.2f}%")

if nb_common > 0:
    print("\nExemples de questions communes :")
    for q in list(intersection)[:5]:
        print("-", q)
### END TODO ###
```

```
Nombre de questions communes entre (train+val) et test : 3
Pourcentage de fuite potentielle : 0.09%

Exemples de questions communes :
- when did mickey mantle play for the yankees
- what year was michael irvin drafted
- who was the president after jfk died
```

Partie 2 - Module d'évaluation (4 pts)

La section suivante comprend l'implémentation d'un module d'évaluation. Nous allons utiliser la métrique `Exact Match` (EM) pour comparer la réponse générée à la réponse attendue. Cela signifie que la réponse sera évaluée comme correcte si et seulement si elle est exactement équivalente à la réponse attendue. Nous allons également utiliser le `Partial Match` (PM), qui dans notre cas sera défini ainsi : la réponse prédite et la réponse attendues sont un partial match si au moins un **mot** est présent à la fois dans la réponse prédite et dans la réponse attendue. Dans les deux cas, nous retirerons la ponctuation et convertirons les deux représentations textuelles en minuscules.

Exemples :

Réponse attendue : Barack Obama

Réponse prédite 1 : Barack Obama --> EM = True, PM = True

Réponse prédite 2 : B. Obama --> EM = False, PM = True

Réponse prédite 3 : George Bush --> EM = False, PM = False

En utilisant ces méthodes, nous allons pouvoir calculer la précision, rappel et F1 par question, étant donnée que la sortie attendue est une liste de réponses à la question.

Q2.1 (1 pts) - Implémentez la fonction suivante qui prend en entrée une string et retourne en sortie cette même string, mise en minuscule et dont la ponctuation a été retirée.

In [ ]:
```python
def normalize_text(text: str) -> str:
    """Minuscule + suppression ponctuation simple"""
    ### TODO ###
    if not isinstance(text, str):
        return ""

    # Mise en minuscule
    text = text.lower()

    # Suppression de la ponctuation (tout ce qui n'est pas lettre, chiffre ou espace)
    text = re.sub(r"[^\w\s]", "", text)

    # Retrait des espaces multiples
    text = re.sub(r"\s+", " ", text).strip()
    ### END TODO ###



    return text
```

Q2.2 (1 pts) - Implémentez la fonction suivante qui prend en entrée deux strings et la méthode à utilser ("exact" pour EM, "partial" pour PM), et retourne en sortie `True` si les deux strings sont un match selon la méthode et `False` si elle ne le sont pas, en appliquant `normalize_text` en premier.

In [ ]:
```python
def compare_entities(str1: str, str2: str, method: str = "exact") -> bool:
    """
    Compare deux entités.
    str1 : première string à comparer
    str2 : deuxième string à comparer
    method: "exact" pour correspondance exacte, "partial" pour correspondance partiel
le
    """
    ### TODO ###
    # Normalisation des textes
    str1_norm = normalize_text(str1)
    str2_norm = normalize_text(str2)

    # Si l'une des deux chaînes est vide
    if not str1_norm or not str2_norm:
        return False

    if method == "exact":
        # Exact Match (égalité parfaite)
        return str1_norm == str2_norm

    elif method == "partial":
        # Partial Match : au moins un mot en commun
        words1 = set(str1_norm.split())
        words2 = set(str2_norm.split())
        return bool(words1.intersection(words2))

    else:
        raise ValueError("La méthode doit être 'exact' ou 'partial'.")
    ### END TODO ###
```

La cellule suivante fournit la classe BaseEvaluator qui utilise les fonctions des deux questions précédentes, afin d'implémenter la logique d'évaluation qui s'assure de la validité des sorties du modèle, et du calcul des métriques.

Q 2.3 (2 pts) Implémentez la méthode `evaluate` qui retourne la précision, le rappel et le f1 score.

In [ ]:

```python
# @title
class BaseEvaluator:
    def __init__(self, df: pd.DataFrame, truth_col: str = "natural_lang_answers", pre
d_col: str = "model_response"):
        self.df = df.copy()
        self.truth_col = truth_col
        self.pred_col = pred_col

    @staticmethod
    def _normalize_text(text: str) -> str:
        """Minuscule + suppression ponctuation simple"""
        return normalize_text(text)

    @staticmethod
    def _parse_list_string(list_str: str) -> List[str]:
        """
        Parse a string representation of a list, handling apostrophes and quotes.

        Args:
            list_str: String that looks like a Python list (e.g., "['item1', 'item
2']")

        Returns:
            List of parsed string items
        """
        import ast

        stripped = list_str.strip()

        # Check if it looks like a Python list
        if not (stripped.startswith('[') and stripped.endswith(']')):
            return []

        # Remove outer brackets
        inner = stripped[1:-1].strip()

        # Handle empty list
        if not inner:
            return []

        # Try ast.literal_eval first (safer and handles apostrophes)
        try:
            parsed = ast.literal_eval(list_str)
            if isinstance(parsed, list):
                return [str(v) for v in parsed]
            else:
                return [str(parsed)]
        except (SyntaxError, ValueError):
            # Manual parsing fallback for malformed strings
            items = []
            current_item = []
            in_quotes = False
            quote_char = None

            for char in inner:
                if char in ('"', "'") and (not in_quotes or char == quote_char):
                    if in_quotes:
                        in_quotes = False
                        quote_char = None
                    else:
                        in_quotes = True
                        quote_char = char
                elif char == ',' and not in_quotes:
```

```
                    if current_item:
                        item_str = ''.join(current_item).strip().strip('"').strip
("'")
                        if item_str:
                            items.append(item_str)
                    current_item = []
                else:
                    current_item.append(char)

            # Add last item
            if current_item:
                item_str = ''.join(current_item).strip().strip('"').strip("'")
                if item_str:
                    items.append(item_str)

            return items

    def _to_list(self, value: Union[str, List[str]]) -> List[str]:
        """Convertit string/list en liste de réponses normalisées"""
        if isinstance(value, str):
            try:
                parsed_items = self._parse_list_string(value)
                return [self._normalize_text(item) for item in parsed_items]
            except Exception as e:
                print(f"Failed to parse '{value}': {e}")
                return []

        elif isinstance(value, list):
            return [self._normalize_text(str(v)) for v in value]

        else:
            print(f"Unexpected type for value: {type(value)}")
            return []


    def _compare_entities(self, reference_entity: str, other_entity: str, method: str
= "exact") -> bool:
        """
        Compare deux entités.
        method: "exact" pour correspondance exacte, "partial" pour correspondance par
tielle
        """
        return compare_entities(reference_entity, other_entity, method)


    def evaluate(self, criteria: str):
        """
        Évalue les prédictions selon le critère donné (exact ou partial).
        """

        precisions, recalls, f1s = [], [], []

        ### TODO ###
        for _, row in self.df.iterrows():
            truths = self._to_list(row[self.truth_col])
            raw_pred = str(row[self.pred_col])
            preds = self._to_list(raw_pred)

            if not preds and raw_pred.strip() and raw_pred.lower() != 'nan':
                # On sépare par virgule si demandé par le prompt, sinon on prend tout
                if ',' in raw_pred:
                    preds = [self._normalize_text(p) for p in raw_pred.split(',')]
                else:
```

```
                preds = [self._normalize_text(raw_pred)]

            if not truths and not preds:
                precisions.append(1.0); recalls.append(1.0); f1s.append(1.0)
                continue
            elif not truths:
                precisions.append(0.0); recalls.append(0.0); f1s.append(0.0)
                continue
            elif not preds:
                precisions.append(0.0); recalls.append(0.0); f1s.append(0.0)
                continue

            correct = 0
            matched_preds = set()

            for truth in truths:
                for i, pred in enumerate(preds):
                    if i in matched_preds:
                        continue
                    # Comparaison selon le critère (exact ou partial)
                    if self._compare_entities(truth, pred, method=criteria):
                        correct += 1
                        matched_preds.add(i)
                        break

            # Calcul des scores pour cette question
            precision = correct / len(preds) if preds else 0.0
            recall = correct / len(truths) if truths else 0.0

            if (precision + recall) > 0:
                f1 = (2 * precision * recall) / (precision + recall)
            else:
                f1 = 0.0

            precisions.append(precision)
            recalls.append(recall)
            f1s.append(f1)
        ### END TODO ###
        # Ajouter les métriques par question au DataFrame
        self.df[f"precision_{criteria}"] = precisions
        self.df[f"recall_{criteria}"] = recalls
        self.df[f"f1_{criteria}"] = f1s

        # Calculer les moyennes des métriques
        avg_precision = np.mean(precisions)
        avg_recall = np.mean(recalls)
        avg_f1 = np.mean(f1s)

        return {
            "avg_precision": avg_precision,
            "avg_recall": avg_recall,
            "avg_f1": avg_f1,
            "df": self.df
        }

    def evaluate_exact_partial(self, id=False):
        """
        Évalue les prédictions avec les critères 'exact' et 'partial'.
        Retourne le dataframe évalué avec toutes les métriques.
        """
        results = {}

        for criteria in ["exact", "partial"]:
```

```
            result = self.evaluate(criteria=criteria)
            results[criteria] = result
            # Keep updating df with both exact and partial metrics
            self.df = result['df']

        if not id:

          # Display results in a formatted table
          print('\n')
          print(f"{'Metric':<15} {'Exact':<15} {'Partial':<15}")
          print("-" * 45)
          print(f"{'Precision':<15} {results['exact']['avg_precision']:.2f}{'':<11}
{results['partial']['avg_precision']:.2f}")
          print(f"{'Recall':<15} {results['exact']['avg_recall']:.2f}{'':<11} {result
s['partial']['avg_recall']:.2f}")
          print(f"{'F1':<15} {results['exact']['avg_f1']:.2f}{'':<11} {results['parti
al']['avg_f1']:.2f}")

        else:
          # Display results in a formatted table
          print('\n')
          print(f"{'Metric':<15} {'Exact':<15}")
          print("-" * 45)
          print(f"{'Precision':<15} {results['exact']['avg_precision']:.2f}{'':<11}")
          print(f"{'Recall':<15} {results['exact']['avg_recall']:.2f}{'':<11}")
          print(f"{'F1':<15} {results['exact']['avg_f1']:.2f}{'':<11}")

        return self.df  # return df with both exact and partial metrics
```

Partie 3 - Génération avec mémoire paramétrique (20 pts)

Maintenant que nous avons le code nécéssaire à l'évaluation et que nous nous sommes familiarisés avec les données, nous allons pouvoir passer à l'utilisation du modèle pour tenter de répondre aux questions. Dans cette section, nous allons simplement interroger le modèle en nous appuyant sur sa **mémoire paramétrique**, c'est à dire toute les capacités et connaissances que le modèle obtient après le pré-entrainement.

3.1 - Instanciation du modèle (1 pts)

Page du modèle (https://huggingface.co/meta-llama/Llama-3.2-3B)

La cellule suivante vous demande d'entrer votre token Huggingface (assurez vous de ne pas l'écrire directement dans le code)

In [ ]:
```
#@ Huggingface Token
hf_token = getpass("Enter your Hugging Face token: ")
#my_token=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
login(token=hf_token, new_session=True)
```

```
Enter your Hugging Face token: ·········
```

La cellule suivante télécharge le modèle depuis Huggingface et configure la quantization (https://huggingface.co/docs/optimum/en/concept_guides/quantization) pour réduire la taille du modèle en mémoire.

Vous devez configurer la quantization (`BitsAndBytesConfig`), instancier le tokenizer (`AutoTokenizer.from_pretrained`) et le modèle en utilisant `AutoModelForCausalLM.from_pretrained` .

Q 3.1.1 (1 pts) Éxécutez la cellule suivante pour télécharger le modèle.

In [ ]:

```python
# Define the model ID
model_id = "meta-llama/Llama-3.2-3B-Instruct"

# Configure quantization - 4-bit quantization for T4 GPU
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
)

### TODO ###
print("Loading tokenizer...")
tokenizer = AutoTokenizer.from_pretrained(model_id, use_fast=True)

# Ajouter un token de padding si absent
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({"pad_token": tokenizer.eos_token})

print("Loading quantized model...")
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=bnb_config,
    device_map="auto"
)

# Adapter la taille des embeddings si on a ajouté un pad_token
model.resize_token_embeddings(len(tokenizer))

### END TODO ###

print("Model loaded successfully!")
print(f"Model device: {next(model.parameters()).device}")
print(f"Model dtype: {next(model.parameters()).dtype}")
```

```
Loading tokenizer...

/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarni
ng:
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 (http
s://huggingface.co/settings/tokens), set it as secret in your Google Colab and resta
rt 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 m
odels or datasets.
  warnings.warn(



Loading quantized model...




Model loaded successfully!
Model device: cuda:0
Model dtype: torch.float16
```

3.2 Fonction de génération (1 pts)

La cellule suivante définit le `pipeline` et la fonction `generate_response` permettant la génération.

In [ ]:
```
pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    dtype=torch.bfloat16,
    device_map="auto",
)

def generate_response(pipe, messages, max_new_tokens=256, temperature=0.7, top_p=0.9,
                      repetition_penalty=1.2, top_k=50, do_sample=True) -> str:
    """
    Generate response using the pipeline with configurable parameters.

    Args:
        pipe: The HuggingFace pipeline
        messages: List of message dictionaries like this : [{"role":user, "content":t
ext}]
        max_new_tokens: Maximum number of tokens to generate
        temperature: Sampling temperature (higher = more random)
        top_p: Nucleus sampling parameter
        repetition_penalty: Penalty for repeating tokens
        top_k: Top-k sampling parameter
        do_sample: Whether to use sampling

    Returns:
        Generated text response
    """
    outputs = pipe(
        messages,
        max_new_tokens=max_new_tokens,
        temperature=temperature,
        top_p=top_p,
        repetition_penalty=repetition_penalty,
        top_k=top_k,
        do_sample=do_sample
    )
    return outputs[0]["generated_text"][-1]["content"]
```

```
Device set to use cuda:0
```

Q 3.1.1 (1 pts) - Imaginez un exemple de question et générez la réponse du modèle à l'aide de la fonction `generate_response` .

In [ ]:
```
### TODO ###
question = "What is the main period of the French Revolution"
messages = [
    {"role": "user", "content": question}
]
response=generate_response(pipe, messages, max_new_tokens=256, temperature=0.7, top_p
=0.9,
                       repetition_penalty=1.2, top_k=50, do_sample=True)

### END TODO ###

print("Question:", question)
print("Model response:", response)
```

```
Question: What is the main period of the French Revolution
Model response: The main period of the French Revolution is generally considered to
be from September 1792, when King Louis XVI was executed by guillotine, marking the
end of the monarchy and the beginning of the First Republic, until July 1794. This t
ime frame encompasses several significant events:

1. The Fall of the Bastille (July 14, 1789): A symbol of royal authority, its captur
e marked the start of the revolution.
2. The Declaration of the Rights of Man and Citizen (August 26, 1789): Established f
undamental human rights in France.
3. The Reign of Terror (September 1793 - July 1794): Characterized by extreme violen
ce, executions, and purges under the Committee of Public Safety.

This initial phase set the stage for further radical changes that would shape modern
France's history. However, some historians also consider the broader movement as spa
nning from the Estates-General call in May 1789 to Napoleon Bonaparte's rise to powe
r in November 1800. Nevertheless, the core event-driven years are typically pinpoint
ed between September 1792 and July 1794.
```

3.3 Prompting (9 pts)

Dans cette section vous allez tenter de trouver un prompt (system + user) permettant d'obtenir des réponses de qualité du modèle.

Dans un cadre de type chat avec un modèle comme Llama 3.2-3B, le system prompt (rôle « system ») sert à donner des instructions globales : par exemple définir le comportement, le style ou les contraintes de l'assistant (« tu es un assistant poli qui répond brièvement », « tu parles comme un professeur »). Le user prompt (rôle « user ») est ce que l'utilisateur demande à l'assistant (la question, la tâche, le contexte particulier). Lorsqu'on combine les deux, le modèle tente de concilier les consignes générales du système avec la requête spécifique de l'utilisateur. Plus de détails ici (https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2/?utm_source=chatgpt.com)

Q3.3.1 (1 pts) - Utilisez le modèle pour répondre à la question 2619 du jeu de données de validation. Passez uniquement la question au modèle :

```
messages = [
    {"role": "user", "content": question}
]
```

Affichez la question `question`, la réponse attendue en ID `original_answers` et en langue naturelle `natural_lang_answers` ainsi que la réponse générée en langue naturelle.

In [ ]:
```
picked_question = dataset['val']['question'].to_list()[2619]
```

In [ ]:
```
### TODO ###
print("Question: ",picked_question)
original_answers=dataset['val']['original_answers'].to_list()[2619]
print("Réponse attendue en ID: ",original_answers)
natural_lang_answers=dataset['val']['natural_lang_answers'].to_list()[2619]
print("Réponse attendue en Natural language: ",natural_lang_answers)
messages = [
    {"role": "user", "content": picked_question}
]
response=generate_response(pipe, messages,max_new_tokens=256, temperature=0.7, top_p=0.9,
                   repetition_penalty=1.2, top_k=50, do_sample=True)
print("Réponse du modèle:")
print(response)
### END TODO ###
```

```
Question:  the september 11 attacks were carried out with the involvement of what te
rrorist organizations
Réponse attendue en ID:  ['Q34490']
Réponse attendue en Natural language:  ['Al-Qaeda']
Réponse du modèle:
The September 11 attacks were carried out by al-Qaeda, a terrorist organization foun
ded by Osama bin Laden. Al-Qaeda is responsible for several other terrorist acts aro
und the world, including the 1998 United States embassy bombings and the 2002 Bali b
ombings.

Additionally, there was also evidence suggesting that the Islamic Jihad Union (IJU)
may have had some level of support or knowledge about the plot, but they did not dir
ectly participate in it.

It's worth noting that while these two groups are primarily associated with the 9/11
attacks, other groups such as Hamas, Hezbollah, and others have been implicated in v
arious ways through funding, training, planning, logistics etc however no concrete e
vidence has linked them to the 9/11 act itself
```

Q 3.3.2 (1 pts) Qu'observez vous par rapport à ce qui est retourné par le modèle en comparaison avec la réponse attendue?

Le modèle a correctement identifié Al-Qaeda comme l'organisation terroriste impliquée dans les attentats du 11 septembre, ce qui correspond à la réponse attendue (Al-Qaeda). Cependant, plusieurs points sont à noter :

Verbosité excessive : La réponse est très détaillée. En effet, elle inclue une explication de la création de l'organisation par Osama ben Laden, ses implications dans plusieurs actes terroristes ainsi que l'implication potentielle d'autres groupes dans l'organisation des attentats du 11 septembre

Tendance à l'hallucination partielle : Le modèle évoque la participation du groupe IJU (Islamic Jihad Union) dans les attentats du 11 septembre alors que cette organisation a été fondée en 2002.

Nous allons donc essayer de trouver un System prompt permettant d'obtenir plus précisément ce que nous cherchons, c'est à dire une réponse directe, et dans le bon format. Pour ce faire, nous allons créer un **`PromptManager`** qui va enregistrer différent system prompts et créer le prompt final pour le modèle.

Nous avons remarqué précédemment que les réponses dans le jeux de données se présentent à la fois sous forme d'identifiant wikidata (ID) et de réponse en langue naturelle (NL).

Q 3.3.3 (2 pts) Complétez la classe `PromptManager` en remplissant les deux types de prompts, l'un demandant au modèle de répondre avec un identifiant, l'autre en langue naturelle. Assurez vous que le prompt que vous créez encourage le modèle a bien répondre (format, etc.).

Q 3.3.4 (2 pts) Complétez la classe `PromptManager` en implémentant la fonction `_get_few_shot_examples`. Cette fonction doit retourner une liste de `num_shots` exemples (question, réponse) tirés aléatoirement du dataframe `train_df`. La réponse doit être dans le format demandé par `response_type` (soit 'natural_language' soit 'entity_ids'). Ces `num_shots` exemples doivent être sélectionnés au hasard parmis les 10 exemples fournis (les `uniq_ids` sont donnés).

Q 3.3.5 (2 pts) Complétez la classe `PromptManager` en implémentant la fonction `create_prompt`. Cette fonction doit créer la liste de messages `system`, `user`, ou `assistant` en incluant les exemples éventuels. Attention les exemples sont inclus dans des messages séparés, avec la question dans le message `user` et la réponse dans le message `assistant`. Ajoutez la phrase suivante au prompt avant les examples si ceux-ci sont inclus : `Here are a few examples:`.

In [ ]:

```
@dataclass
class PromptManager():

    basic: str = """You are a helpful question answering assistant.
Answer the question based on the provided information.
Provide only the answer in a clear and concise format."""

    # Q3.3.3 - Réponse en langue naturelle
    # Objectif : réponse directe, courte, sans explications inutiles.
    standard_nl: str = """You are an answer-only QA model. Your goal is to output exa
ctly one canonical natural-language answer for each question. Follow the rules strict
ly.

Rules:
Output only one single answer, even if the question is plural or mentions multiple en
tities
Output the most canonical, commonly accepted natural-language answer.
Do not output lists, commas, "and", "or", or multiple items.
Do not explain anything. Do not repeat the question.
Keep punctuation only if it is part of the canonical name. Do not add periods.
For years or numbers, output only the number."""

    # Q3.3.3 - Réponse en identifiants Wikidata
    # Objectif : renvoyer uniquement les IDs (Qxxxx), rien d'autre.
    standard_id: str = """You are a question answering assistant.
Answer ONLY with the Wikidata entity ID(s) (e.g., Q42).
Do not output any words, explanations, or additional text.
If there are multiple correct entities, output all IDs separated by a space.
If you do not know, answer: "none"."""

    prompts = {
        "basic": basic,
        "standard_nl": standard_nl,
        "standard_id": standard_id
    }

    def _get_few_shot_examples(self, train_df: Any, num_shots: int, response_type: st
r) -> List[Tuple[str, str]]:
        few_shot_ids = [3043, 5696, 6149, 988, 1045, 6889, 7074, 7363, 7544, 9642]
        resp_type = response_type.lower()

        # Filter dataframe
        candidates = train_df[train_df['id'].isin(few_shot_ids)]
        if candidates.empty or num_shots <= 0:
            return []

        # Sample
        num_shots = min(num_shots, len(candidates))
        sampled_rows = random.sample(candidates.to_dict('records'), num_shots)

        examples: List[Tuple[str, str]] = []

        for row in sampled_rows:
            q = str(row['question'])

            # Define source column and separator based on type
            if resp_type in ["nl", "natural_language"]:
                raw = row['natural_lang_answers']
                separator = ", "
            else:
                raw = row['original_answers']
                separator = " "
```

```
            a = str(raw) # Default fallback

            # Attempt to parse list strings
            if isinstance(raw, str) and raw.strip().startswith('['):
                try:
                    parsed = ast.literal_eval(raw)
                    if isinstance(parsed, list) and len(parsed) > 0:
                        a = separator.join(str(x) for x in parsed)
                    elif not isinstance(parsed, list):
                        a = str(parsed)
                except Exception:
                    pass # Keep default 'a'

            examples.append((q, a))

        return examples
        ### END TODO ###

    def create_prompt(self, query: str, prompt_type: str, response_type: str = "nl",
                      num_shots: int = 0, train_df: Optional[Any] = None) -> List[Dic
t[str, str]]:
        """
        Create a base prompt with the specified configuration and optional few-shot e
xamples.
        Args:
            prompt_type: Type of prompt to use (e.g., 'standard_id', 'basic', 'standa
rd_nl')
            query: The user's question
            num_shots: Number of few-shot examples to include (0 for zero-shot)
            train_df: Training dataframe for few-shot examples
            response_type: 'nl'/'natural_language' or 'entity_ids'
        Returns:
            List of message dictionaries for chat format
        """
        ### TODO ###
        if prompt_type not in self.prompts:
            raise ValueError(f"Unknown prompt_type: {prompt_type}")

        # Récupérer le bon system prompt
        system_prompt = self.prompts[prompt_type]

        # Message système de base
        messages: List[Dict[str, str]] = [
            {"role": "system", "content": system_prompt}
        ]

        # Ajouter des few-shot examples si demandés
        if num_shots > 0 and train_df is not None:
            examples = self._get_few_shot_examples(train_df, num_shots, response_typ
e)
            if examples:
                # Phrase demandée avant les exemples
                messages.append({
                    "role": "system",
                    "content": "Here are a few examples:"
                })
                for q, a in examples:
                    messages.append({"role": "user", "content": q})
                    messages.append({"role": "assistant", "content": a})

        # Ajouter la vraie question de l'utilisateur
        messages.append({"role": "user", "content": query})
        ### END TODO ###
```

```
        return messages

    def display_prompt_template(self, messages: List[Dict[str, str]]) -> None:
        """
        Display the formatted prompt template using tokenizer.apply_chat_template
        Args:
            messages: List of message dictionaries to display
        """
        formatted_prompt = tokenizer.apply_chat_template(messages, tokenize=False)
        print("Formatted prompt:\n")
        print(formatted_prompt)
```

La cellule suivante utilise un `PromptManager` pour créer le prompt et permet d'afficher le [chat_template (https://huggingface.co/docs/transformers/main/chat_templating)](https://huggingface.co/docs/transformers/main/chat_templating) utilisé par le modèle.

In [ ]:
```
pm = PromptManager()

prompt = pm.create_prompt(
    query=picked_question,
    prompt_type="standard_nl"
)

pm.display_prompt_template(prompt)

response = generate_response(
    pipe,
    prompt
)
#print(f"\nResponse:\n{first_response_nl}\n")
print(f"Model response:\n{response}")
```

```
Formatted prompt:

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 30 Nov 2025

You are an answer-only QA model. Your goal is to output exactly one canonical natura
l-language answer for each question. Follow the rules strictly.

Rules:
Output only one single answer, even if the question is plural or mentions multiple e
ntities
Output the most canonical, commonly accepted natural-language answer.
Do not output lists, commas, "and", "or", or multiple items.
Do not explain anything. Do not repeat the question.
Keep punctuation only if it is part of the canonical name. Do not add periods.
For years or numbers, output only the number.<|eot_id|><|start_header_id|>user<|end_
header_id|>

the september 11 attacks were carried out with the involvement of what terrorist org
anizations<|eot_id|>
Model response:
al-Qaeda and the Islamic Jihad Organization
```

Q 3.3.6 (1 pts) En vous basant sur la cellule ci-dessus, détaillez ce que fait `apply_chat_template` pour ce modèle.

La méthode apply_chat_template permet de convertir une liste de messages en un format compréhensible par un modèle linguistique. Cette function formate ainsi une entrée structurée (comme une liste de dictionnaires avec « role » et « content ») en une invite spécifique pour le modèle. Elle structure ainsi les interactions les utilisateurs un modèle de langue.

La cellule suivante définit des fonctions utiles pour créer des prompts et générer des réponses par batch. Vous devez implémenter les fonctions `create_all_prompts` et `generate_all_responses_batch` .

`create_all_prompts` permet de créer tous les prompts d'un dataset donné et retourne une liste de prompts. Lorsque nous voudrons passer à travers toutes les questions d'un `split` donné, nous devrons d'abord créer la liste de prompts, puis générer les réponses par batch grâce à la fonction `generate_all_responses_batch` . Ainsi vous utiliserez `batch_outputs = pipe(...)` avec le bon `batch_size` .

In [ ]:

```python
def create_all_prompts(dataset_df: pd.DataFrame,
                       prompt_manager: PromptManager,
                       prompt_type: str = "standard_id",
                       num_shots: int = 0,
                       train_df: Optional[pd.DataFrame] = None) -> List[List[Dict[str, str]]]:
    """
    Create prompts for all questions in the dataset.

    Args:
        dataset_df: Original dataset dataframe
        prompt_manager: Instance of PromptManager
        prompt_type: Type of prompt to use
        response_type: Type of response expected
        num_shots: Number of few-shot examples
        train_df: Training dataframe for few-shot examples

    Returns:
        List of prompts, where each prompt is a list of message dictionaries
    """
    print(f"Creating prompts for {len(dataset_df)} questions...")
    questions = dataset_df['question'].to_list()
    all_prompts = []

    ### TODO ###
    # Déduire le type de réponse attendu à partir du type de prompt
    if "id" in prompt_type.lower():
        response_type = "entity_ids"
    else:
        response_type = "nl"

    # Créer un prompt par question
    for q in questions:
        prompt = prompt_manager.create_prompt(
            query=q,
            prompt_type=prompt_type,
            response_type=response_type,
            num_shots=num_shots,
            train_df=train_df
        )
        all_prompts.append(prompt)
    ### END TODO ###

    print(f"Created {len(all_prompts)} prompts")
    return all_prompts

def generate_all_responses_batch(pipe,
                                 prompts: List[List[Dict[str, str]]],
                                 batch_size: int = 8,
                                 max_new_tokens: int = 64,
                                 temperature: float = 0.3,
                                 top_p: float = 0.8,
                                 **generation_kwargs) -> List[str]:
    """
    Generate responses for a list of prompts using pipeline batching.

    Args:
        pipe: The HuggingFace pipeline
        prompts: List of prompts, where each prompt is a list of message dictionaries
        batch_size: Maximum number of samples in a batch
        max_new_tokens: Maximum number of tokens to generate
        temperature: Sampling temperature (higher = more random)
        top_p: Nucleus sampling parameter
```

```
    Returns:
        List of generated responses
    """
    print(f"Generating responses for {len(prompts)} prompts in batches of {batch_siz
e}")

    all_responses = []

    ### TODO ###
    # Parcours par batch
    for i in range(0, len(prompts), batch_size):
        batch_prompts = prompts[i:i + batch_size]

        batch_outputs = pipe(
            batch_prompts,
            max_new_tokens=max_new_tokens,
            temperature=temperature,
            top_p=top_p,
            **generation_kwargs
        )

        # batch_outputs est une liste, chaque entrée correspond à un prompt
        for out in batch_outputs:
            # Compatible avec le format utilisé dans generate_response(...)
            # - cas le plus fréquent: out[0]["generated_text"][-1]["content"]
            if isinstance(out, list):
                gen = out[0]["generated_text"][-1]["content"]
            else:
                gen = out["generated_text"][-1]["content"]
            all_responses.append(gen)
    ### END TODO ###

    print(f"Generated {len(all_responses)} responses")
    return all_responses

def create_response_dataframe(dataset_df: pd.DataFrame,
                              responses: List[str],
                              config: Dict[str, Any] = None) -> pd.DataFrame:
    """
    Create a dataframe with original data and model responses.

    Args:
        dataset_df: Original dataset dataframe
        responses: List of generated responses
        generation_config: Configuration used for generation

    Returns:
        DataFrame with original data plus model responses and metadata
    """
    df_response = dataset_df.copy()
    df_response['model_response'] = responses

    # Add metadata about the prompting and generation process
    if config:
        for key, value in config.items():
            df_response[f'config_{key}'] = value

    return df_response
```

3.4 Génération des réponses (1 pts)

Q 3.4.1 (1 pts) - La cellule suivante permet d'exécuter la génération du modèle sur l'ensemble de validation. Vous pouvez changer le type de prompt en changeant la variable `prompt_type` du dictionnaire `prompting_config` . Exécutez cette cellule pour les deux types de prompt (nl et id). Les résultats seront enregistrés dans le dossier 'cache' pour que vous puissiez y référer ensuite.

In [ ]:

```
prompting_config = {
    'prompt_type': 'standard_nl',# a run pour 'prompt_type': 'standard_id' de base st
andard_nl
}

generation_config = {
    'max_new_tokens': 128,
    'temperature': 0.3,
    'top_p': 0.8,
    'batch_size': 16
}

full_config = {**prompting_config, **generation_config}
cache_file_name = f"{DATA_PATH}/cache/val_data_model_response_{prompting_config['prom
pt_type'].split('_')[1]}.csv"

if os.path.exists(cache_file_name):
    print("Loading cached responses...")
    df_response = pd.read_csv(cache_file_name, quoting=1)
    print("Cached responses loaded successfully!")

else:
    # Generate prompts for val dataset
    print("Generating prompts for val dataset...")
    val_prompts = create_all_prompts(
        dataset_df=dataset['val'],
        prompt_manager=pm,
        prompt_type=prompting_config["prompt_type"]
    )

    # Generate responses for val dataset
    print("Generating responses for val dataset...")
    val_responses = generate_all_responses_batch(
        pipe=pipe,
        prompts=val_prompts,
        **generation_config
    )

    # Create response dataframe
    df_response = create_response_dataframe(
        dataset_df=dataset['val'],
        responses=val_responses,
        config=full_config
    )

    print(f"Generated {len(val_responses)} responses")
    df_response.head()

    # Save with proper quoting to prevent future errors
    df_response.to_csv(cache_file_name, index=False, quoting=1)
    print(f"Saved responses to {cache_file_name}")
```

```
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Generating responses for val dataset...
Generating responses for 3236 prompts in batches of 16

You seem to be using the pipelines sequentially on GPU. In order to maximize efficie
ncy please use a dataset

Generated 3236 responses
Generated 3236 responses
Saved responses to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_model_r
esponse_nl.csv
```

3.5 Evaluation (4 pts)

Q 3.5.1 (1 pts) - À l'aide des fonctions et classes définies précédemment, évaluez la performance des modèles à prédire les réponses en langue naturelle et en identifiant sur le set de validation en utilisant les metriques (EM, PM). Analysez la différence de performance entre les deux approches.

In [ ]:

```
# Enregistrez les réponses de cette manière pour en pas avoir à ré-éxécuter la généra
tion (lorsque vous aurez trouvé un prompt satisfaisant)
"""cache_file_name = f"{DATA_PATH}/cache/file_name.csv"

if not os.path.exists(cache_file_name):
    raise FileNotFoundError(f"Cache file not found: {cache_file_name}")

df_response_nl = pd.read_csv(cache_file_name)
print("Cached responses loaded successfully!")"""

### TODO ###
# Load cache
cache_file_name_nl = f"{DATA_PATH}/cache/val_data_model_response_nl.csv"
if os.path.exists(cache_file_name_nl):
    df_response_nl = pd.read_csv(cache_file_name_nl)
    print("Cached NL responses loaded successfully!")
else:
    print(f"NL cache file not found: {cache_file_name_nl}")
    df_response_nl = None

# Run Evaluation
if df_response_nl is not None:
    print("\nEvaluating Natural Language Responses:")
    evaluator_nl = BaseEvaluator(df_response_nl, truth_col="natural_lang_answers", pr
ed_col="model_response")
    evaluator_nl.evaluate_exact_partial()
else:
    print("\nSkipping NL evaluation due to missing cache file.")


### END TODO ###
```

```
Cached NL responses loaded successfully!

Evaluating Natural Language Responses:


Metric          Exact           Partial
-----------------------------------------------
Precision       0.12            0.34
Recall          0.10            0.30
F1              0.10            0.30
```

In [ ]:
```python
### TODO ###
#Load cache
cache_file_id = f"{DATA_PATH}/cache/val_data_model_response_id.csv"
if os.path.exists(cache_file_id):
    df_response_id = pd.read_csv(cache_file_id)
    print("Cached ID responses loaded successfully!")
else:
    print(f"ID cache file not found: {cache_file_id}")
    df_response_id = None

# Run Evaluation
if df_response_id is not None:
    print("\nEvaluating Entity ID Responses:")
    evaluator_id = BaseEvaluator(df_response_id, truth_col="original_answers", pred_c
ol="model_response")
    evaluator_id.evaluate_exact_partial(id=True) # Use id=True for ID evaluation
else:
    print("\nSkipping ID evaluation due to missing cache file.")
### END TODO ###
```

```
Cached ID responses loaded successfully!

Evaluating Entity ID Responses:


Metric          Exact
-----------------------------------------------
Precision       0.00
Recall          0.00
F1              0.00
```

Q 3.5.2 (2 pts) Donnez vos observations sur les différences de performance entre les deux types de réponses (ID vs NL). S'il y en a une, expliquez la différence de performance.

Résultat obtenu suite aux réponses sous forme de langage naturel:

	$Exact$	$Partial$
$Precision$	0.12	0.34
$Recall$	0.10	0.30
$F1$	0.12	0.30

Résultat obtenu suite aux réponses sous forme d'ID Wikidata:

	$Exact$
$Precision$	0.00
$Recall$	0.00
$F1$	0.00

Nous pouvons remarquer que les deux types de performances (ID et NL) obtiennent des scores de précision, de recall et de f1 différents pour la métrique Exact Match. En effet, selon le score Exact Match, les métriques sous forme de langage obtiennent des scores légerement supérieur à 0.1 alors les métriques sous ID Wikidata restent bloquer à 0.

Ainsi, le modèle se montre plus performant lorsqu'il répond en langage naturel. Cette différence s'explique principalement par la nature des données d'entraînement. En effet, le modèle a été entraîné sur du texte en langage naturel, ce qui lui permet de mieux s'adapter à des tâches variées lié au lagange naturel. N'ayant pas été entraîné spécifiquement sur les identifiants Wikidata, il lui est très difficile de produire la réponse exacte attendue dans ce format.

Par conclusion, cet écart de performance entre les métriques Exact Match s'explique par la nature des données d'entraînement qui ont favorisé la réponse sous forme de langage naturel.

Q 3.5.2 (1 pts) - Retournez à la question Q 3.3.3 et changez les prompts que vous avez créé jusqu'à obtention d'un score d'au moins 0.1 F1-Score sur l'ensemble de validation avec les réponses sous forme de langue naturelle en exact match. Cela est important pour pouvoir poursuivre la suite du travail.

3.6 Exemples Few-shot (4 pts)

Afin d'améliorer les performances, nous allons maintenant intégrer des exemples au prompt, en utilisant le few-shot prompting (https://huggingface.co/docs/transformers/tasks/prompting#few-shot-prompting). La classe PromptManager permet déja cette intégration simplement.

Q 3.6.1 (1 pts) - Testez la réponse du LLM sur la question choisie précédemment en utilisant un prompt avec 5 exemples (shots).

In [ ]:
```python
### TODO ###
prompt_with_few_shots = pm.create_prompt(
    query=picked_question,
    prompt_type="standard_nl",
    num_shots=5,
    train_df=dataset['train']
)

response_with_few_shots = generate_response(
    pipe,
    prompt_with_few_shots
)

print("Question:", picked_question)
print("Model response with 5 few-shot examples:", response_with_few_shots)

### END TODO ###
```

```
Question: the september 11 attacks were carried out with the involvement of what ter
rorist organizations
Model response with 5 few-shot examples: ['Al-Qaeda and Islamic Jihad']
```

Q 3.6.2 (1 pts) - Évaluez les performances du modèle avec 5 exemples sur tout le dataset de validation.

Vous devez gérer la `prompting_config` et la `generation_config` en vous aidant de la question Q 3.4.1. Nous voudrions avoir 5 exemples provenant du train set. Nous vous conseillons d'utiliser un cache file pour enregistrer vos réponse. Vous devez passer au travers du set de validation en appellant les trois fonctions outils : `create_all_prompts` , `generate_all_responses_batch` et `create_response_dataframe` .

In [ ]:

```
#@title Génération
### TODO ###
prompting_config = {
    'prompt_type': 'standard_nl',
    'num_shots': 5 # Ensure num_shots is set to 5
}
generation_config = {
    'max_new_tokens': 128,
    'temperature': 0.3,
    'top_p': 0.8,
    'batch_size': 16
}

full_config = {**prompting_config, **generation_config}
cache_file_name = f"{DATA_PATH}/cache/val_data_model_5_response_{prompting_config['pr
ompt_type'].split('_')[1]}.csv"

if os.path.exists(cache_file_name):
    print("Loading cached responses...")
    df_response = pd.read_csv(cache_file_name, quoting=1)
    print("Cached responses loaded successfully!")

else:
    # Generate prompts for val dataset
    print("Generating prompts for val dataset...")
    val_prompts = create_all_prompts(
        dataset_df=dataset['val'],
        prompt_manager=pm,
        prompt_type=prompting_config["prompt_type"],
        num_shots=prompting_config["num_shots"], # Pass num_shots from config
        train_df=dataset['train'] # Pass train dataset for few-shot examples
    )

    # Generate responses for val dataset
    print("Generating responses for val dataset...")
    val_responses = generate_all_responses_batch(
        pipe=pipe,
        prompts=val_prompts,
        **generation_config
    )

    # Create response dataframe
    df_response = create_response_dataframe(
        dataset_df=dataset['val'],
        responses=val_responses,
        config=full_config
    )

    print(f"Generated {len(val_responses)} responses")
    df_response.head()

    # Save with proper quoting to prevent future errors
    df_response.to_csv(cache_file_name, index=False, quoting=1)
    print(f"Saved responses to {cache_file_name}")

### END TODO ###
```

```
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Generating responses for val dataset...
Generating responses for 3236 prompts in batches of 16

You seem to be using the pipelines sequentially on GPU. In order to maximize efficie
ncy please use a dataset

Generated 3236 responses
Generated 3236 responses
Saved responses to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_model_5
_response_nl.csv
```

In [ ]:
```python
#@title Évaluation
### TODO ###
cache_file_name_nl = f"{DATA_PATH}/cache/val_data_model_5_response_nl.csv"
if os.path.exists(cache_file_name_nl):
    df_response_nl = pd.read_csv(cache_file_name_nl)
    print("Cached NL num_shots responses loaded successfully!")
else:
    print(f"NL num_shots cache file not found: {cache_file_name_nl}")
    df_response_nl = None

# Run Evaluation
if df_response_nl is not None:
    print("\nEvaluating Natural Language Responses for num_shots:")
    evaluator_nl = BaseEvaluator(df_response_nl, truth_col="natural_lang_answers", pr
ed_col="model_response")
    evaluator_nl.evaluate_exact_partial()
else:
    print("\nSkipping NL num_shots evaluation due to missing cache file.")

### END TODO ###
```

```
Cached NL num_shots responses loaded successfully!

Evaluating Natural Language Responses for num_shots:


Metric          Exact           Partial
-----------------------------------------------
Precision       0.16            0.31
Recall          0.14            0.26
F1              0.14            0.27
```

Q 3.6.3 (2 pts) Commentez et expliquez l'impact du few-shot prompting sur les performances du modèle.

Résultat obtenu suite aux réponses sous forme de langage naturel sans few-shots:

	$Exact$	$Partial$
$Precision$	0.12	0.34
$Recall$	0.10	0.30
$F1$	0.12	0.30

Résultat obtenu suite aux réponses sous forme de langage naturel avec few-shots:

	$Exact$	$Partial$
$Precision$	0.16	0.31
$Recall$	0.14	0.26
$F1$	0.14	0.27

Nous pouvons remarquer que le few-shot prompting a eu un impact sur les performances du modèle. En effet, l'ensemble des métriques pour le score Exact et Partial sont supérieurs à ceux obtenus précédemment sans exemples. Cette amélioration s'explique notamment par la nature du few-shot prompting. En effet, cette méthode consiste à fournir au modèle plusieurs exemples représentatifs de la tâche avant de lui demander une nouvelle prédiction.

Ces exemples permettent ainsi de guider le modèle en lui faisant comprendre la structure attendue des réponses. En fournissant ces exemples, on donne au modèle un cadre struturé lui permettant de raisonner et d'être ainsi aligné avec les réponses attendues.

Ainsi, le few-shot prompting agit comme un tuteur qui par sa présence améliore la qualité de ses réponses.

Partie 4 - Retrieval Augmented Generation (14 pts)

Les résultats de la dernière section devraient motiver l'utilisation d'une technique populaire et vue en classe : Retrieval Augmented Generation (https://huggingface.co/docs/transformers/en/model_doc/rag) (RAG)

Dans cette partie nous allons implémenter la classe `CustomRetriever` et la fonction `batch_retrieve` pour créer un système de recherche de documents (retrieval) optimisé qui trouve les documents (faits) les plus pertinents pour une requête (question) donnée.

Concernant les exigences fonctionnelles, nous allons:

- Utiliser le modèle d'embeddings **"all-MiniLM-L6-v2"** de HuggingFace
- Calculer les embeddings de tous les documents à l'initialisation
- Implémenter la similarité cosinus pour mesurer la pertinence
- Retourner les top-k documents les plus similaires à la question

4.1 Retriever

Nous allons tout d'abord créer une classe `CustomRetriever` qui sera responsable de créer les embeddings et retrouver les éléments les plus similaires.

Q 4.1.1 (1 pts) - Complétez le code de la fonction `compute_document_embeddings_optimized()` qui, de manière optimisée avec traitement par batch, calcule les embeddings de tous les documents. Cette fonction sera réutilisée dans le `CustomRetriever` .

In [ ]:
```
def compute_document_embeddings_optimized(documents, embedding_model, batch_size=16):
    """
    Cette méthode traite les documents par groupes (batches) pour optimiser
    les performances et réduire l'overhead des appels au modèle d'embedding.
    La taille des batches est définie par self.batch_size.
    Input:
    documents: Liste de documents (faits)
    embedding_model: Instance du modèle d'embedding à utiliser
    batch_size: Taille des batches pour le traitement
    Returns:
        np.ndarray: Array NumPy 2D contenant les embeddings de tous les documents.
                  Shape: (n_documents, embedding_dimension)
                  L'ordre des embeddings correspond à l'ordre des documents
                  dans self.documents
    """
    embeddings = []
    for i in range(0, len(documents), batch_size):# création des batchs
        batch_documents = documents[i:i + batch_size]
        for docucment in batch_documents:
            try:
                doc_dict = json.loads(docucment.replace("'", '"'))
                text = f"{doc_dict.get('subject', '')} {doc_dict.get('relation', '')}
{doc_dict.get('object', '')}" #only the text
                processed_text = text.strip()
            except json.JSONDecodeError:
                processed_text = str(docucment).strip()

            embedding = embedding_model.get_text_embedding(processed_text)# calcul em
beddings
            embeddings.append(embedding)

    return np.array(embeddings)
```

In [ ]:

```
class CustomRetriever:
    def __init__(self, documents, embed_model_name="all-MiniLM-L6-v2", cache_file="em
beddings_cache.pkl", batch_size=32, n_workers=4):
        """
        Initialiser le récupérateur personnalisé optimisé avec des embeddings

        Args:
            documents: Liste de documents (faits)
            embed_model_name: Nom du modèle d'embedding à utiliser
            cache_file: Fichier de cache pour les embeddings
            batch_size: Taille des batches pour le traitement
            n_workers: Nombre de threads pour le parallélisme
        """
        self.documents = documents
        self.embed_model_name = embed_model_name
        self.cache_file = cache_file
        self.batch_size = batch_size
        self.n_workers = n_workers

        # Utilisation des embeddings HuggingFace
        self.embed_model = HuggingFaceEmbedding(
            model_name=self.embed_model_name,
            trust_remote_code=True
        )

        # Essayer de charger depuis le cache d'abord
        if self._load_embeddings_from_cache():
            print(f"\tEmbeddings chargés depuis le cache : {self.cache_file}")
        else:
            print(f"\tCalcul des embeddings pour {len(documents)} documents avec {sel
f.embed_model_name}...")
            self.document_embeddings = self._compute_document_embeddings_optimized()
            self._save_embeddings_to_cache()
            print(f"\tEmbeddings sauvegardés dans le cache : {self.cache_file}")

    def _load_embeddings_from_cache(self):
        """Charger les embeddings depuis le cache"""
        if os.path.exists(self.cache_file):
            try:
                with open(self.cache_file, 'rb') as f:
                    cache_data = pickle.load(f)

                # Vérifier si le cache correspond aux documents actuels
                if (cache_data['documents'] == self.documents and
                    cache_data['model_name'] == self.embed_model_name):
                    self.document_embeddings = cache_data['embeddings']
                    return True
            except:
                print("\tErreur lors du chargement du cache, recalcul des embedding
s...")
        return False

    def _save_embeddings_to_cache(self):
        """Sauvegarder les embeddings dans le cache"""
        cache_data = {
            'documents': self.documents,
            'model_name': self.embed_model_name,
            'embeddings': self.document_embeddings
        }
        with open(self.cache_file, 'wb') as f:
            pickle.dump(cache_data, f)

    def _compute_document_embeddings_optimized(self):
```

```
        """
        Cette méthode traite les documents par groupes (batches) pour optimiser
        les performances et réduire l'overhead des appels au modèle d'embedding.
        La taille des batches est définie par self.batch_size.

        Returns:
            np.ndarray: Array NumPy 2D contenant les embeddings de tous les document
s.
                        Shape: (n_documents, embedding_dimension)
                        L'ordre des embeddings correspond à l'ordre des documents
                        dans self.documents

        """

        return compute_document_embeddings_optimized(self.documents, self.embed_mode
l, batch_size=self.batch_size)


    def retrieve(self, query, top_k=3):
        """
        Récupérer les top-k documents les plus pertinents pour une requête
        Version optimisée avec cache des query embeddings
        """
        # Cache simple pour les requêtes (optionnel)
        if not hasattr(self, '_query_cache'):
            self._query_cache = {}

        if query in self._query_cache:
            query_embedding = self._query_cache[query]
        else:
            query_embedding = self.embed_model.get_text_embedding(query)
            query_embedding = np.array(query_embedding).reshape(1, -1)
            self._query_cache[query] = query_embedding

        # Calculer la similarité cosinus
        similarities = cosine_similarity(query_embedding, self.document_embeddings)
[0]

        # Obtenir les top-k documents les plus similaires
        top_indices = np.argsort(similarities)[::-1][:top_k]

        retrieved_docs = []
        for idx in top_indices:
            doc_obj = type('Document', (), {'text': self.documents[idx]})()
            retrieved_docs.append(doc_obj)

        return retrieved_docs
```

La cellule suivante crée les plongements pour les documents (faits) qui seront utilisés pour le RAG.

In [ ]:
```
# Utilisation optimisée
print(" Initialisation du Custom Retriever Optimisé...")
documents_list = facts_dataframe['fact'].to_list()

# Avec cache automatique
retriever = CustomRetriever(
    documents_list,
    cache_file=f"{DATA_PATH}/cache/embeddings_cache.pkl", # Use a unique cache file n
ame for the subset
    batch_size=16  # Augmenter si vous avez assez de RAM
)

print("\tRetriever initialisé et prêt à l'usage !")
```

```
 Initialisation du Custom Retriever Optimisé...
```

```
        Embeddings chargés depuis le cache : /content/drive/MyDrive/NLP/TP4_Amaury/d
ata/cache/embeddings_cache.pkl
        Retriever initialisé et prêt à l'usage !
```

Q 4.1.2 (1 pts) - Pour la même question sélectionnée en Q 3.3.1, utilisez le retriever pour trouver les document pertinents et affichez-les.

In [ ]:
```
picked_question = picked_question

### TODO ###
print(f"Question: {picked_question}")
retrieved_docs = retriever.retrieve(picked_question, top_k=3)
print("Documents retrouvés:")
for idx, doc in enumerate(retrieved_docs):
    print(f"{idx + 1}. {doc.text}")
    print("-" * 50)

### END TODO ###
```

```
Question: the september 11 attacks were carried out with the involvement of what ter
rorist organizations
Documents retrouvés:
1. {'subject': 'The CIA and September 11', 'relation': 'main subject', 'object': 'Se
ptember 11 attacks'}
--------------------------------------------------
2. {'subject': 'September 11 attacks', 'relation': 'instance of', 'object': 'terrori
st attack'}
--------------------------------------------------
3. {'subject': 'September 11 attacks', 'relation': 'participant', 'object': 'Al-Qaed
a'}
--------------------------------------------------
```

Q 4.1.3 (3 pts) - Implémentez la fonction `batch_retrieve(questions, retriever, batch_size=64)` qui effectue une recherche (retrieval) optimisée en lot pour plusieurs questions simultanément, en vectorisant les calculs de similarité pour améliorer les performances.

In [ ]:

```python
def batch_retrieve(questions, retriever, batch_size=64, top_k=3):
    """
    Effectue un retrieval en batch pour plusieurs questions de manière optimisée
    Cette fonction traite plusieurs questions simultanément en calculant tous les
    query embeddings en batch, puis en utilisant la vectorisation NumPy pour
    calculer les similarités de toutes les questions en une seule opération.
    Args:
        questions (List[str]): Liste des questions à traiter
        retriever (CustomRetriever): Instance du retriever avec embeddings pré-calcul
és
        batch_size (int, optional): Taille des batches pour le traitement.
                                    Defaults to 64.
    Returns:
        Tuple[List[List[int]], List[List[str]]]: Tuple contenant:
            - all_retrieved_facts_ids: Liste des listes d'IDs des top-3 faits pour ch
aque question
            - all_retrieved_facts_labels: Liste des listes de textes des top-3 faits
pour chaque question
    """
    print(f"Computing embeddings for {len(questions)} queries...")
    ### TODO ###
    query_embeddings = []
    # Calcul de tous les embeddings des questions
    for question in questions:
        embedding = retriever.embed_model.get_text_embedding(question) # embed questi
ons
        query_embeddings.append(embedding)

    query_embeddings = np.array(query_embeddings)

    all_retrieved_facts_ids = []
    all_retrieved_facts_labels = []


    if not isinstance(retriever.document_embeddings, np.ndarray):
      # Transformation en array pour le calcul de similarité
        retriever.document_embeddings = np.array(retriever.document_embeddings)

    # Traitement par batch
    for i in range(0, len(questions), batch_size):
        batch_query_embeddings = query_embeddings[i:i + batch_size]

        # Similarités entre batch de questions et tous les documents
        similarities = cosine_similarity(batch_query_embeddings, retriever.document_e
mbeddings)

        # Récupération des indices des top_k documents
        top_indices = np.argsort(similarities, axis=1)[:, ::-1][:, :top_k]

        batch_retrieved_facts_ids = top_indices.tolist()

        # Récupération des labels/textes correspondants
        batch_retrieved_facts_labels = []
        for indices_per_question in top_indices:
            labels_for_question = [retriever.documents[idx] for idx in indices_per_qu
estion]
            batch_retrieved_facts_labels.append(labels_for_question) # ajout aux list
e

        all_retrieved_facts_ids.extend(batch_retrieved_facts_ids)
        all_retrieved_facts_labels.extend(batch_retrieved_facts_labels)

    return all_retrieved_facts_ids, all_retrieved_facts_labels
```

Q 4.1.4 (1 pts) Application de la recherche de documents (retrieval)

Pour toutes les questions du jeu de validation, faite la recherche de document pertinents en utilisant le retriever instancié précedemment. Les identifiants et les textes des documents retrouvés seront ensuite ajoutés au dataframe du jeu de test nommé `df_val` .

In [ ]:
```
### TODO ###
### TODO ###
# Extraction des questions du dataset de validation
all_questions = dataset['val']['question'].to_list()

#Récupération des documents pertinents par batch
all_retrieved_facts_ids, all_retrieved_facts_labels = batch_retrieve(all_questions, r
etriever, batch_size=64, top_k=3)

#Ajout des colonnes
dataset['val']['retrieved_facts_ID'] = all_retrieved_facts_ids
dataset['val']['retrieved_facts_Text'] = all_retrieved_facts_labels

### END TODO ###
### END TODO ###
```

```
Computing embeddings for 3236 queries...
```

Q 4.1.5 (1 pts) La cellule suivante définit une fonction qui permet d'ajouter les faits qui ont été retrouvés au prompt de génération. Vous devez implémenter la fonction. Ajoutez `'Here is some information to help you answering the question:'` au dernier message user avant d'inclure les faits.

In [ ]:
```python
def add_facts_to_prompt(self,messages: List[Dict[str, str]], retrieved_facts: List[str]) -> List[Dict[str, str]]:
    """
    Adds retrieved facts to an existing prompt message list.

    Args:
        messages: The existing list of message dictionaries.
        retrieved_facts: List of retrieved facts to include.

    Returns:
        List of message dictionaries with facts added to the last user message.
    """
    ### TODO ###
    # On cherche le dernier message 'user'
    last_user_message_index = -1
    for i in range(len(messages) - 1, -1, -1):
        if messages[i]["role"] == "user":
            last_user_message_index = i
            break

    # Ajout des faits
    if last_user_message_index != -1:
        facts_text = "\nHere is some information to help you answering the question:\n" + "\n".join(retrieved_facts)
        messages[last_user_message_index]["content"] += facts_text
    else:
        messages.append({"role": "user", "content": "Here is some information to help you answering the question:\n" + "\n".join(retrieved_facts)})

    ### END TODO ###

    return messages
```

Q 4.1.6 (1 pts) Afficher la réponse générée pour la même `picked_question` de la Q 3.3.1 en utilisant `add_facts_to_prompt` .

In [ ]:
```python
# Récupérer les documents correspondant à la question sélectionnée
retrieved_docs = retriever.retrieve(picked_question, top_k=3)

# Convertir les documents récupérés en une liste de chaînes
retrieved_facts_text = [doc.text for doc in retrieved_docs]

# Création du prompt
initial_prompt = pm.create_prompt(
    query=picked_question,
    prompt_type="standard_nl"
)

# Ajout des faits au prompt
prompt_with_facts = add_facts_to_prompt(pm,
    initial_prompt,
    retrieved_facts_text
)

# Génerer la réponse avec la prise en compte des faits
response = generate_response(pipe, prompt_with_facts)

print("Question:", picked_question)
print("Model response with facts:", response)
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.

Question: the september 11 attacks were carried out with the involvement of what ter
rorist organizations
Model response with facts: Al-Qaeda
```

4.2 Évaluation

Les 3 cellules suivantes donnent du code permettant d'exécuter le RAG avec différentes configurations. Assurez-vous de bien le comprendre afin de pouvoir l'utiliser.

In [ ]:

```python
def run_rag_evaluation(dataset, retriever, pm, pipe, top_k_values=[1, 3, 5],
                       prompting_config=None, generation_config=None,
                       data_path=DATA_PATH, split="val"):
    """
    Exécute l'évaluation RAG pour plusieurs valeurs de top_k

    Args:
        dataset: Dictionnaire contenant les splits du dataset
        retriever: Instance du retriever
        pm: PromptManager instance
        pipe: Pipeline de génération
        top_k_values: Liste des valeurs de top_k à tester
        prompting_config: Configuration du prompting
        generation_config: Configuration de la génération
        data_path: Chemin vers le dossier data
        split: Donne le split à utiliser (train, val ou test)

    Returns:
        Dict contenant les résultats pour chaque top_k
    """

    id_ = False

    if prompting_config is None:
        prompting_config = {
            'prompt_type': 'standard_id',
            'response_type': 'entity_ids',
            'num_shots': 0
        }

    if generation_config is None:
        generation_config = {
            'max_new_tokens': 128,
            'temperature': 0.3,
            'top_p': 0.8,
            'batch_size': 16
        }

    # Déterminer la colonne de vérité
    if prompting_config['prompt_type'] == 'standard_id':
        truth_col = "original_answers"
        id_ = True
    elif prompting_config['prompt_type'] == 'standard_nl':
        truth_col = "natural_lang_answers"
        id_=False

    results = {}

    for top_k in top_k_values:
        print(f"\n{'='*80}")
        print(f"EVALUATION WITH RETRIEVED FACTS (TOP_K = {top_k})")
        print(f"{'='*80}\n")

        # 1. Préparation des facts
        retrieval_cache_file = f"{data_path}/cache/{split}_data_retrieved_fact_cache_
top{top_k}.csv"

        if os.path.exists(retrieval_cache_file):
            print(f"Loading cached retrieved facts (top-{top_k})...")
            df_split = pd.read_csv(retrieval_cache_file)
        else:
            print(f"Running batch_retrieve with top_k={top_k}...")
            df_split = dataset[split].copy()
```

```
            results_ids, results_texts = batch_retrieve(
                df_split['question'].to_list(),
                retriever,
                batch_size=16,
                top_k=top_k
            )
            df_split['retrieved_facts_ID'] = results_ids
            df_split['retrieved_facts_Text'] = results_texts
            df_split.to_csv(retrieval_cache_file, index=False)
            print(f"Saved to {retrieval_cache_file}")

        # 2. Génération avec RAG
        facts_type = "retrieved"

        response_cache_file = f"{data_path}/cache/{split}_data_model_response_{facts_
type}_top{top_k}_{prompting_config['prompt_type'].split('_')[1]}.csv"

        if os.path.exists(response_cache_file):
            print(f"Loading cached responses (top-{top_k})...")
            df_response = pd.read_csv(response_cache_file)
        else:
            print(f"Generating prompts for {split} dataset...")
            split_prompts = create_all_prompts(
                dataset_df=dataset[split],
                prompt_manager=pm,
                **prompting_config
            )

            print(f"Adding top-{top_k} facts to prompts...")
            for i, row in df_split.iterrows():
                split_prompts[i] = add_facts_to_prompt(
                    pm,
                    split_prompts[i],
                    row['retrieved_facts_Text']
                )

            print(f"Generating responses...")
            split_responses = generate_all_responses_batch(
                pipe=pipe,
                prompts=split_prompts,
                **generation_config
            )

            full_config = {**prompting_config, **generation_config, 'top_k': top_k,
'facts_type': facts_type}
            df_response = create_response_dataframe(
                dataset_df=dataset[split],
                responses=split_responses,
                config=full_config
            )
            df_response.to_csv(response_cache_file, index=False)
            print(f"Saved to {response_cache_file}")

        # 3. Évaluation
        print(f"\nEvaluating with top_k={top_k}...")
        evaluator = BaseEvaluator(df_response, truth_col=truth_col, pred_col="model_r
esponse")
        df_evaluated = evaluator.evaluate_exact_partial(id=id_)

        # Stocker les résultats
        results[top_k] = {
            'df_evaluated': df_evaluated,
            'exact_precision': df_evaluated['precision_exact'].mean(),
```

```
            'exact_recall': df_evaluated['recall_exact'].mean(),
            'exact_f1': df_evaluated['f1_exact'].mean(),
            'partial_precision': df_evaluated['precision_partial'].mean(),
            'partial_recall': df_evaluated['recall_partial'].mean(),
            'partial_f1': df_evaluated['f1_partial'].mean()
        }

    return results
```

In [ ]:

```
def display_comparison_results(results):
    """
    Affiche un tableau comparatif des résultats pour différentes valeurs de top_k

    Args:
        results: Dictionnaire retourné par run_rag_evaluation
    """
    print(f"\n{'='*100}")
    print("COMPARISON OF RESULTS ACROSS DIFFERENT TOP_K VALUES")
    print(f"{'='*100}\n")

    # En-tête
    header = f"{'Top_K':<10}"
    for metric in ['Precision', 'Recall', 'F1']:
        header += f"{'Exact ' + metric:<20}{'Partial ' + metric:<20}"
    print(header)
    print("-" * 100)

    # Lignes de données
    for top_k in sorted(results.keys()):
        row = f"{top_k:<10}"
        row += f"{results[top_k]['exact_precision']:.4f}{'':<15}"
        row += f"{results[top_k]['partial_precision']:.4f}{'':<15}"
        row += f"{results[top_k]['exact_recall']:.4f}{'':<15}"
        row += f"{results[top_k]['partial_recall']:.4f}{'':<15}"
        row += f"{results[top_k]['exact_f1']:.4f}{'':<15}"
        row += f"{results[top_k]['partial_f1']:.4f}{'':<15}"
        print(row)

    print("\n" + "="*100)

    # Trouver la meilleure valeur de top_k
    best_top_k_exact = max(results.keys(), key=lambda k: results[k]['exact_f1'])
    best_top_k_partial = max(results.keys(), key=lambda k: results[k]['partial_f1'])

    print(f"\n Best top_k for Exact Match F1: {best_top_k_exact} (F1={results[best_to
p_k_exact]['exact_f1']:.4f})")
    print(f" Best top_k for Partial Match F1: {best_top_k_partial} (F1={results[best_
top_k_partial]['partial_f1']:.4f})")
```

In [ ]:
```python
def plot_comparison_results(results):
    """
    Crée des graphiques de comparaison pour les différentes valeurs de top_k

    Args:
        results: Dictionnaire retourné par run_rag_evaluation
    """
    top_k_values = sorted(results.keys())

    # Préparer les données
    exact_precision = [results[k]['exact_precision'] for k in top_k_values]
    exact_recall = [results[k]['exact_recall'] for k in top_k_values]
    exact_f1 = [results[k]['exact_f1'] for k in top_k_values]

    partial_precision = [results[k]['partial_precision'] for k in top_k_values]
    partial_recall = [results[k]['partial_recall'] for k in top_k_values]
    partial_f1 = [results[k]['partial_f1'] for k in top_k_values]

    # Créer les graphiques
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))

    # Exact Match
    axes[0].plot(top_k_values, exact_precision, marker='o', label='Precision', linewi
dth=2)
    axes[0].plot(top_k_values, exact_recall, marker='s', label='Recall', linewidth=2)
    axes[0].plot(top_k_values, exact_f1, marker='^', label='F1', linewidth=2)
    axes[0].set_xlabel('Top K', fontsize=12)
    axes[0].set_ylabel('Score', fontsize=12)
    axes[0].set_title('Exact Match Metrics vs Top K', fontsize=14, fontweight='bold')
    axes[0].legend(fontsize=11)
    axes[0].grid(True, alpha=0.3)
    axes[0].set_xticks(top_k_values)

    # Partial Match
    axes[1].plot(top_k_values, partial_precision, marker='o', label='Precision', line
width=2)
    axes[1].plot(top_k_values, partial_recall, marker='s', label='Recall', linewidth=
2)
    axes[1].plot(top_k_values, partial_f1, marker='^', label='F1', linewidth=2)
    axes[1].set_xlabel('Top K', fontsize=12)
    axes[1].set_ylabel('Score', fontsize=12)
    axes[1].set_title('Partial Match Metrics vs Top K', fontsize=14, fontweight='bol
d')
    axes[1].legend(fontsize=11)
    axes[1].grid(True, alpha=0.3)
    axes[1].set_xticks(top_k_values)

    plt.tight_layout()
    plt.show()
```

Q 4.2.1 (3 pts) - Testez maintenant le pipeline de RAG avec quatre valeurs de k (représentant le nombre de faits à inclure) : 1, 3, 5 et 10 sur l'ensemble de validation, en utilisant les fonctions `run_rag_evaluation` , `display_comparison_results` et `plot_comparison_results` . Il faudra faire l'evaluation pour les deux types de réponses: `standard_nl` et `standard_id` .

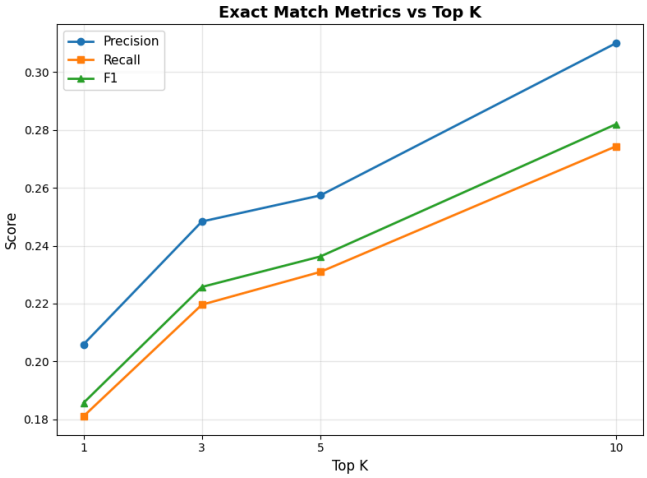
In [ ]:

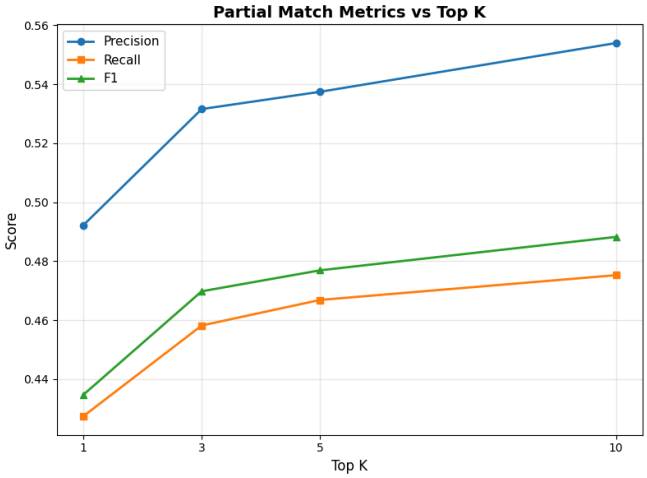
```
# Définir les valeurs de top_k à tester
TOP_K_VALUES = [1, 3, 5, 10]
pm = PromptManager()

retriever = CustomRetriever(
    documents_list,
    cache_file=f"{DATA_PATH}/cache/embeddings_cache.pkl", # Use a unique cache file n
ame for the subset
    batch_size=16  # Augmenter si vous avez assez de RAM
)
prompt = pm.create_prompt(
    query=picked_question,
    prompt_type="standard_nl"
)

generation_config = {
    'max_new_tokens': 128,
    'temperature': 0.3,
    'top_p': 0.8,
    'batch_size': 16
}
pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    dtype=torch.bfloat16,
    device_map="auto",
)

### TODO ###
print("Running RAG evaluation for Natural Language responses...")
prompting_config_nl = {
    'prompt_type': 'standard_nl',
}
results_nl = run_rag_evaluation(
    dataset=dataset,
    retriever=retriever,
    pm=pm,
    pipe=pipe,
    top_k_values=TOP_K_VALUES,
    prompting_config=prompting_config_nl,
    generation_config=generation_config,
    data_path=DATA_PATH,
    split="val"
)
display_comparison_results(results_nl)
plot_comparison_results(results_nl)



### END TODO ###
```

```
Device set to use cuda:0

        Embeddings chargés depuis le cache : /content/drive/MyDrive/NLP/TP4_Amaury/d
ata/cache/embeddings_cache.pkl
Running RAG evaluation for Natural Language responses...

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 1)
================================================================================

Running batch_retrieve with top_k=1...
Computing embeddings for 3236 queries...
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_retrieved_fact_ca
che_top1.csv
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Adding top-1 facts to prompts...
Generating responses...
Generating responses for 3236 prompts in batches of 16

You seem to be using the pipelines sequentially on GPU. In order to maximize efficie
ncy please use a dataset
```

```
Generated 3236 responses
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_model_response_re
trieved_top1_nl.csv

Evaluating with top_k=1...


Metric          Exact           Partial
-----------------------------------------
Precision       0.21            0.49
Recall          0.18            0.43
F1              0.19            0.43

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 3)
================================================================================

Running batch_retrieve with top_k=3...
Computing embeddings for 3236 queries...
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_retrieved_fact_ca
che_top3.csv
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Adding top-3 facts to prompts...
Generating responses...
Generating responses for 3236 prompts in batches of 16
Generated 3236 responses
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_model_response_re
trieved_top3_nl.csv

Evaluating with top_k=3...


Metric          Exact           Partial
-----------------------------------------
Precision       0.25            0.53
Recall          0.22            0.46
F1              0.23            0.47

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 5)
================================================================================

Running batch_retrieve with top_k=5...
Computing embeddings for 3236 queries...
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_retrieved_fact_ca
che_top5.csv
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Adding top-5 facts to prompts...
Generating responses...
Generating responses for 3236 prompts in batches of 16
Generated 3236 responses
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_model_response_re
trieved_top5_nl.csv

Evaluating with top_k=5...


Metric          Exact           Partial
-----------------------------------------
```

```
Precision       0.26            0.54
Recall          0.23            0.47
F1              0.24            0.48

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 10)
================================================================================

Running batch_retrieve with top_k=10...
Computing embeddings for 3236 queries...
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_retrieved_fact_ca
che_top10.csv
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Adding top-10 facts to prompts...
Generating responses...
Generating responses for 3236 prompts in batches of 16
Generated 3236 responses
Saved to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/val_data_model_response_re
trieved_top10_nl.csv

Evaluating with top_k=10...


Metric          Exact           Partial
--------------------------------------------
Precision       0.31            0.55
Recall          0.27            0.48
F1              0.28            0.49

================================================================================
================
COMPARISON OF RESULTS ACROSS DIFFERENT TOP_K VALUES
================================================================================
================

Top_K     Exact Precision     Partial Precision   Exact Recall        Partial Recall
Exact F1            Partial F1
--------------------------------------------------------------------------------
----------------
1         0.2059              0.4922              0.1811              0.4274
0.1857              0.4347
3         0.2484              0.5316              0.2196              0.4582
0.2257              0.4698
5         0.2574              0.5374              0.2309              0.4668
0.2363              0.4769
10        0.3100              0.5540              0.2744              0.4752
0.2820              0.4882

================================================================================
================

 Best top_k for Exact Match F1: 10 (F1=0.2820)
 Best top_k for Partial Match F1: 10 (F1=0.4882)
```


Exact Match Metrics vs Top K
Precision
Recall
F1
Score
Top K
0.18
0.20
0.22
0.24
0.26
0.28
0.30
1
3
5
10


Partial Match Metrics vs Top K
Precision
Recall
F1
Score
Top K
0.44
0.46
0.48
0.50
0.52
0.54
0.56
1
3
5
10

In [ ]:

```
print("\nRunning RAG evaluation for Entity ID responses...")
TOP_K_VALUES = [1, 3, 5, 10]
pm = PromptManager()
retriever = CustomRetriever(
    documents_list,
    cache_file=f"{DATA_PATH}/cache/embeddings_cache.pkl", # Use a unique cache file n
ame for the subset
    batch_size=16  # Augmenter si vous avez assez de RAM
)
prompting_config_id = {
    'prompt_type': 'standard_id',
}
generation_config = {
    'max_new_tokens': 128,
    'temperature': 0.3,
    'top_p': 0.8,
    'batch_size': 16
}
pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    dtype=torch.bfloat16,
    device_map="auto",
)
results_id = run_rag_evaluation(
    dataset=dataset,
    retriever=retriever,
    pm=pm,
    pipe=pipe,
    top_k_values=TOP_K_VALUES,
    prompting_config=prompting_config_id,
    generation_config=generation_config,
    data_path=DATA_PATH,
    split="val"
)
display_comparison_results(results_id)
plot_comparison_results(results_id)
```

```
Running RAG evaluation for Entity ID responses...
Device set to use cuda:0
```

```
        Embeddings chargés depuis le cache : /content/drive/MyDrive/NLP/TP4_Amaury/d
ata/cache/embeddings_cache.pkl

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 1)
================================================================================

Loading cached retrieved facts (top-1)...
Loading cached responses (top-1)...

Evaluating with top_k=1...


Metric          Exact
------------------------------------------
Precision       0.00
Recall          0.00
F1              0.00

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 3)
================================================================================

Loading cached retrieved facts (top-3)...
Loading cached responses (top-3)...

Evaluating with top_k=3...


Metric          Exact
------------------------------------------
Precision       0.00
Recall          0.00
F1              0.00

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 5)
================================================================================

Loading cached retrieved facts (top-5)...
Loading cached responses (top-5)...

Evaluating with top_k=5...


Metric          Exact
------------------------------------------
Precision       0.00
Recall          0.00
F1              0.00

================================================================================
EVALUATION WITH RETRIEVED FACTS (TOP_K = 10)
================================================================================

Loading cached retrieved facts (top-10)...
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Adding top-10 facts to prompts...
Generating responses...
Generating responses for 3236 prompts in batches of 16
```

```
You seem to be using the pipelines sequentially on GPU. In order to maximize efficie
ncy please use a dataset
```

Q 4.2.2 (2 pts) - Observez et commentez la performance du modèle avec RAG pour les différentes valuers de k.

Rappel des résultat avec un simple prompt pour les réponses sous forme de langage naturel:

	$Exact$	$Partial$
$Precision$	0.12	0.34
$Recall$	0.10	0.30
$F1$	0.12	0.30

Rappel des résultat avec la méthode few-shots pour les réponses sous forme de langage naturel :

	$Exact$	$Partial$
$Precision$	0.16	0.31
$Recall$	0.14	0.26
$F1$	0.14	0.27

Dans un premier temps, nous pouvons observer que les métriques (précision, Recall, F1-Score) pour le calcul des Exact Match selon les réponses sous forme de langage naturel suivent une tendance linéaire croissante. L'ajout sucessif des documents permet une amélioration continue des métriques pour le score Exact Match. De plus, cette amélioration continue se confirme en analysant les métriques partial match. En effet, les différentes métriques ne cessent de s'améliorer avec l'ajout des documents.Par exemple, le score F1 selon Partial match augmente de 0.43 pour un document à 0.49 pour 10 documents.

De plus, nous pouvons analyser que la méthode RAG performe mieux que les exemples few-shots précédemment étudiés. Bien que les résultats few-shots ont permis une amélioration des métriques selon le Score Exact, la méthode RAG même avec un seul document performe mieux que les exemples few-shots. Cette augmentation du score démontre ainsi la puissance du RAG quelquesoit le nombre de documents forunis.

Rappel des résultat avec un simple prompt pour les réponses sous forme d'ID Wikidata:

	$Exact$
$Precision$	0.0
$Recall$	0.0
$F1$	0.0

Nous pouvons remarquer que l'ajout de documents pour les réponses sous forme d'ID ne permettent aucune amélioration du score Exact Match par rapport à du simple prompting. En effet, quelques soit le nombre de documents apportés au modèle, les scores restent bloquer à 0 selon la métrique Exact Match.

Q 4.2.3 (1 pts) - Comment expliquez-vous les résultats obtenus ?

Ce gain de performance concernant les réponses sous forme de langage naturel s'explique principalement par la capacité du RAG à fournir des documents pertinents et concrets en lien direct avec la question posée. Contrairement au simple prompting, où le modèle se base uniquement sur ses connaissances internes, le RAG enrichit les connaissances du LLM. Bien que la méthode few-shots apporte quelques exemples génériques de réponses au modèle améliorant ces performances, la méthode RAG apporte des informations concrètes et précises pour l'ensemble des questions lui permettant d'ameliorer sa précision globable.

En apportant des données complémentaires précises pour chaque question, le RAG permet au modèle de "prendre un temps de réflexion" plus structuré. Cela lui permet notamment de conditionner sa réponse non seulement en fonction du prompt initial, mais aussi en tenant compte des documents récupérés et de la requête de l'utilisateur. Cette approche améliore ainsi la précision et la pertinence car elle réduit drastiquement le risque d'hallucination.

Concernant les identifiants Wikidata, l' absence d'amélioration peut notamment s'expliquer par la difficulté de la tâche à laquelle le modèle est confronté. En effet, le modèle doit non seulement comprendre la question de l'utilisateur, mais aussi transformer la réponse sous forme d'ID Wikidata. Bien que les document fournissent des informations de réponses, il ne permmettent pas d'apporter des indices sur la tâche de transformation sous forme d'ID. Ainsi, cette absence d'information concernant les identifiants Wikidata empêche une amélioration consistante du modèle.

Partie 5 : Affinage avec Q-LoRA (12 pts)

Nous allons maintenant affiner directement le modèle avec Q-LoRA (https://huggingface.co/docs/peft/main/en/developer_guides/quantization#qlora-style-training). Les parties suivantes de code sont fournies:

- Prétraitement du jeu de données pour l'entrainement
- Création de la configuration pour QLoRA
- Création de la configuration d'entrainement
- Fonction d'entrainement
- Instanciation du modèle entrainé
- Redémarrage du modèle (ceci peut être utilisé pour débugger votre entrainement)

5.1 Affinage

Q 5.1.1 (3 pts) Prétraitement du jeu de données pour l'entraînement

Pour l'affinage, nous devons adapter les données dans le format Hugging Face approprié. Pour chaque paires de question / réponse du training set, vous devez en faire un `training_example`. Ce dernier est un dictionnaire de 3 éléments : `messages`, `question`, `target_answer`. `messages` est créé en appelant la méthode `create_prompt` du `PromptManager` (doc (https://huggingface.co/docs/transformers/main/en/chat_templating)).

In [ ]:

```python
def prepare_training_data(dataset_dict: dict,
                          prompt_manager,
                          prompt_type: str = "standard_nl",
                          response_type: str = "nl",
                          num_shots: int = 0) -> Dataset:
    """
    Prepare training data in the format needed for fine-tuning.

    Args:
        dataset_dict: Dictionary containing 'train', 'val', 'test' splits
        prompt_manager: Your PromptManager instance
        prompt_type: Type of prompt to use
        response_type: Column name for target answers
        num_shots: Number of few-shot examples (0 for fine-tuning)

    Returns:
        Hugging Face Dataset ready for training
    """

    if response_type == "nl":
        row_name = "natural_lang_answers"
    elif response_type == "id":
        row_name = "original_answers"
    else:
        raise ValueError(f"Unknown response_type: {response_type}")

    train_df = dataset_dict['train']
    training_examples = []

    print(f"Preparing {len(train_df)} training examples...")

    ### TODO ###
    for idx, row in train_df.iterrows():
        question = row["question"]
        target_answer = row[row_name]

        # prompt complet
        messages = prompt_manager.create_prompt(
            query=question,
            prompt_type=prompt_type,
            response_type=response_type,
            num_shots=num_shots
        )

        messages.append({"role": "assistant", "content": str(target_answer)})


        # exemple d'entraînement
        training_example = {
            "messages": messages,
            "question": question,
            "target_answer": target_answer
        }

        training_examples.append(training_example)

    ### END TODO ###

    return Dataset.from_list(training_examples)

def format_chat_template(example, tokenizer):
    """
    Format the conversation using the model's chat template.
```

```
    """
    conversation = example["messages"]
    formatted_text = tokenizer.apply_chat_template(
        conversation,
        tokenize=False,
        add_generation_prompt=False
    )
    return {"text": formatted_text}

def tokenize_function(examples, tokenizer, max_length=512):
    """
    Tokenize the formatted text for training.
    """
    tokenized = tokenizer(
        examples["text"],
        truncation=True,
        padding="max_length",  # Pad to max_length for consistent tensor sizes
        max_length=max_length,
        return_tensors=None
    )
    tokenized["labels"] = tokenized["input_ids"].copy()
    return tokenized
```

Création de la configuration pour QLoRA

In [ ]:
```
def setup_qlora_config():
    """
    Set up the LoRA configuration for Q-LoRA fine-tuning.
    """
    lora_config = LoraConfig(
        task_type=TaskType.CAUSAL_LM,
        r=16,  # Rank of adaptation
        lora_alpha=32,  # LoRA scaling parameter
        lora_dropout=0.1,  # LoRA dropout
        target_modules=[
            "q_proj", "k_proj", "v_proj", "o_proj",
            "gate_proj", "up_proj", "down_proj"
        ],
        bias="none",
    )
    return lora_config
```

Création de la configuration d'entrainement

In [ ]:
```python
def setup_training_args(output_dir: str = "./qlora-results"):
    training_args = TrainingArguments(
        output_dir=output_dir,
        per_device_train_batch_size=4,
        per_device_eval_batch_size=4,
        gradient_accumulation_steps=4,
        num_train_epochs=1,
        learning_rate=2e-4,
        lr_scheduler_type="cosine",
        warmup_steps=100,
        logging_steps=10,

        # === Checkpoints intermédiaires ===
        save_strategy="steps",       # sauvegarder toutes les X steps
        save_steps=50,              # sauvegarde toutes les 50 steps
        save_total_limit=3,          # garder seulement les 3 derniers checkpoints
        eval_strategy="steps",
        eval_steps=50,

        load_best_model_at_end=True,
        metric_for_best_model="eval_loss",
        greater_is_better=False,

        dataloader_pin_memory=False,
        optim="paged_adamw_8bit",
        gradient_checkpointing=True,
        fp16=True,
        report_to="none",
        remove_unused_columns=False,
    )
    return training_args
```

Q 5.1.2 (3 pts) Fonction d'affinage

Vous devez implémenter les 6 étapes ci-dessous avant de lancer l'entraînement. Vous pouvez vous référer aux commentaires des étapes allant de 1 à 6.

In [ ]:

```
def fine_tune_model(model, tokenizer, dataset_dict, prompt_manager, prompt_type: str
= "standard_nl", response_type: str = "nl",
                    num_shots:int =0, output_dir: str = "./qlora-results"):
    """
    Complete fine-tuning pipeline.

    Args:
        model: Your quantized model
        tokenizer: Your tokenizer
        dataset_dict: Dictionary with train/val/test splits
        prompt_manager: Your PromptManager instance
        prompt_type: Type of prompt to use
        output_dir: Where to save the fine-tuned model

    Returns:
        Trained model and trainer
    """

    ### TODO ###

    print("Step 1: Setting up LoRA configuration...")

    lora_config = setup_qlora_config()

    print("Step 2: Applying LoRA to model...")
    model = prepare_model_for_kbit_training(model)
    model = get_peft_model(model, lora_config)

    print("Step 3: Preparing training dataset...")
    train_dataset = prepare_training_data(
        dataset_dict=dataset_dict["train"],
        prompt_manager=prompt_manager,
        prompt_type=prompt_type,
        response_type=response_type,
        num_shots=num_shots
    )

    val_dataset = prepare_training_data(
        dataset_dict=dataset_dict["val"],
        prompt_manager=prompt_manager,
        prompt_type=prompt_type,
        response_type=response_type,
        num_shots=num_shots
    )


    print("Step 4: Formatting and tokenizing datasets...")
    # Apply formatting and tokenization using map
    train_dataset = train_dataset.map(
        lambda x: format_chat_template(x, tokenizer),
        remove_columns=["messages", "question", "target_answer"]
    )
    train_dataset = train_dataset.map(
        lambda x: tokenize_function(x, tokenizer),
        batched=True,
        remove_columns=["text"]
    )

    val_dataset = val_dataset.map(
        lambda x: format_chat_template(x, tokenizer),
        remove_columns=["messages", "question", "target_answer"]
    )
    val_dataset = val_dataset.map(
```

```
        lambda x: tokenize_function(x, tokenizer),
        batched=True,
        remove_columns=["text"]
    )


    print("Step 5: Setting up training arguments...")

    training_args = setup_training_args(output_dir=output_dir)

    print("Step 6: Creating trainer...")

    data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
    trainer = Trainer(
        model=model,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
        tokenizer=tokenizer,
        data_collator=data_collator,
        args=training_args,
    )


    ### END TODO ###

    print("Step 7: Starting training...")
    #trainer.train(resume_from_checkpoint=True)
    try:
        # Look for an existing checkpoint in the output directory
        checkpoints = [d for d in os.listdir(output_dir) if os.path.isdir(os.path.joi
n(output_dir, d)) and d.startswith("checkpoint-")]
        if checkpoints:
            latest_checkpoint = os.path.join(output_dir, sorted(checkpoints, key=lamb
da x: int(x.split('-')[-1]))[-1])
            print(f"Resuming training from checkpoint: {latest_checkpoint}")
            trainer.train(resume_from_checkpoint=latest_checkpoint)
        else:
            print("No checkpoint found — starting training from scratch.")
            trainer.train()
    except ValueError as e:
        # This handles the case where resume_from_checkpoint=True but no checkpoint i
s valid
        print(f"Could not resume from checkpoint ({e}). Starting from scratch instea
d.")
        trainer.train()

    print("Step 8: Saving model...")
    trainer.save_model()

    return model, trainer
```

Instanciation du modèle entraîné

In [ ]:

```python
def load_finetuned_model(base_model_id: str, adapter_path: str):
    """
    Load a fine-tuned Q-LoRA model for inference.

    Args:
        base_model_id: Original model ID (e.g., "meta-llama/Llama-3.2-3B-Instruct")
        adapter_path: Path to saved LoRA adapters

    Returns:
        Loaded model with adapters
    """
    # Load base quantized model
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16,
        bnb_4bit_use_double_quant=True,
    )

    base_model = AutoModelForCausalLM.from_pretrained(
        base_model_id,
        quantization_config=bnb_config,
        device_map="auto",
        dtype=torch.float16,
        trust_remote_code=True
    )

    # Load and apply LoRA adapters
    model = PeftModel.from_pretrained(base_model, adapter_path)

    return model
```

Redémarage du modèle

In [ ]:
```python
def clean_model_reload():
    """
    Clean reload of the model to avoid PEFT conflicts.
    """
    # Clear GPU memory
    torch.cuda.empty_cache()

    # Model configuration
    model_id = "meta-llama/Llama-3.2-3B-Instruct"

    # Configure quantization - 4-bit quantization for T4 GPU
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16,
        bnb_4bit_use_double_quant=True,
    )

    print("Loading tokenizer...")
    tokenizer = AutoTokenizer.from_pretrained(model_id)

    # Add pad token if it doesn't exist
    if tokenizer.pad_token is None:
        tokenizer.padding_side = "left"
        tokenizer.pad_token = tokenizer.eos_token

    print("Loading fresh quantized model...")
    model = AutoModelForCausalLM.from_pretrained(
        model_id,
        quantization_config=bnb_config,
        device_map="auto",
        dtype=torch.float16,
        trust_remote_code=True
    )

    print("Model loaded successfully!")
    print(f"Model device: {next(model.parameters()).device}")
    print(f"Model dtype: {next(model.parameters()).dtype}")

    return model, tokenizer
```

Entraînement

Q 5.1.3 (2 pts) - Exécutez maintenant les deux cellules suivantes pour entraîner le modèle dans les deux configurations (ID et NL) pour une époque. Changez les arguments d'entrainement au besoin.

In [ ]:
```
print("Starting Q-LoRA fine-tuning for NL responses...")
finetuned_model_nl, trainer_nl = fine_tune_model(
    model=model,
    tokenizer=tokenizer,
    dataset_dict=dataset,
    prompt_manager=pm,
    prompt_type="standard_nl",
    response_type="nl",
    output_dir=f"{DATA_PATH}/cache/qlora-nl_answers"
)
```

```
Starting Q-LoRA fine-tuning on ID...
Step 1: Setting up LoRA configuration...
Step 2: Applying LoRA to model...

/usr/local/lib/python3.12/dist-packages/peft/mapping_func.py:73: UserWarning: You ar
e trying to modify a model with PEFT for a second time. If you want to reload the mo
del with a different config, make sure to call `.unload()` before.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/peft/tuners/tuners_utils.py:196: UserWarnin
g: Already found a `peft_config` attribute in the model. This will lead to having mu
ltiple adapters in the model. Make sure to know what you are doing!
  warnings.warn(

Step 3: Preparing training dataset...
Preparing 6970 training examples...
Preparing 6970 training examples...
Step 4: Formatting and tokenizing datasets...



Step 5: Setting up training arguments...
Step 6: Creating trainer...
Step 7: Starting training...
No checkpoint found — starting training from scratch.

/tmp/ipython-input-1814765899.py:76: FutureWarning: `tokenizer` is deprecated and wi
ll be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instea
d.
  trainer = Trainer(
```

[135/436 1:31:13 < 3:26:26, 0.02 it/s, Epoch 0.31/1]

Step	Training Loss	Validation Loss
50	0.232800	0.208946
100	0.135200	0.130638

In [ ]:
```
print("Starting Q-LoRA fine-tuning for ID responses...")
finetuned_model_id, trainer_id = fine_tune_model(
    model=model,
    tokenizer=tokenizer,
    dataset_dict=dataset,
    prompt_manager=pm,
    prompt_type="standard_id",
    response_type="id",
    output_dir=f"{DATA_PATH}/cache/qlora-id_answers"
)
```

```
Starting Q-LoRA fine-tuning on NL...
Step 1: Setting up LoRA configuration...
Step 2: Applying LoRA to model...
Step 3: Preparing training dataset...
Preparing 6970 training examples...
Preparing 6970 training examples...
Step 4: Formatting and tokenizing datasets...



Step 5: Setting up training arguments...
Step 6: Creating trainer...
Step 7: Starting training...

/tmp/ipython-input-1814765899.py:76: FutureWarning: `tokenizer` is deprecated and wi
ll be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instea
d.
  trainer = Trainer(
The tokenizer has new PAD/BOS/EOS tokens that differ from the model config and gener
ation config. The model config and generation config were aligned accordingly, being
updated with the tokenizer's values. Updated tokens: {'eos_token_id': 128009, 'pad_t
oken_id': 128009}.

Resuming training from checkpoint: /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/
qlora-natural_lang_answers/checkpoint-250

`use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fal
se`.
```

[436/436 2:17:01, Epoch 1/1]

Step	Training Loss	Validation Loss
300	0.288100	0.241888
350	0.248300	0.234723
400	0.261600	0.231853

```
Step 8: Saving model...
```

5.2 Évaluation

In [ ]:
```
def eval_fine_tune(finetuned_model, response_path, split = "val", prompt_type: str =
"standard_id", response_type: str = "id"):

  cache_file_name = f"{DATA_PATH}/cache/{response_path}.csv"

  if os.path.exists(cache_file_name):
      print("Loading cached responses...")
      df_response = pd.read_csv(cache_file_name)
      print("Cached responses loaded successfully!")

  else:
      pipe = pipeline(
          "text-generation",
          model=finetuned_model,
          tokenizer=tokenizer,
          dtype=torch.bfloat16,
          device_map="auto",
      )

      prompting_config = {
          'prompt_type': prompt_type,
          'response_type': response_type,
      }

      generation_config = {
          'max_new_tokens': 128,
          'temperature': 0.3,
          'top_p': 0.8,
          'batch_size': 16  # Adjust based on GPU memory
      }

      full_config = {**prompting_config, **generation_config}
      # Generate prompts for split dataset
      print(f"Generating prompts for {split} dataset...")
      prompts = create_all_prompts(
          dataset_df=dataset[split],
          prompt_manager=pm,
          prompt_type=prompting_config["prompt_type"]
      )

      # Generate responses for split dataset
      print(f"Generating responses for {split} dataset...")
      responses = generate_all_responses_batch(
          pipe=pipe,
          prompts=prompts,
          **generation_config
      )

      # Create response dataframe
      df_response = create_response_dataframe(
          dataset_df=dataset[split],
          responses=responses,
          config=full_config
      )

      print(f"Generated {len(responses)} responses")
      df_response.head()
```

```
        df_response.to_csv(cache_file_name, index=False)
        print(f"Saved responses to {cache_file_name}")

    return df_response
```

Q 5.2.1 (2 pts) Pour les deux configurations, chargez le modèle affiné en utilisant load_finetuned_model, puis évaluez-le sur le set de validation en utilisant la fonction `eval_fine_tune()` .

In [ ]:

```
### TODO ###

base_model_id = "meta-llama/Llama-3.2-3B-Instruct"
adapter_path_nl = f"{DATA_PATH}/cache/qlora-natural_lang_answers/checkpoint-436" # As
suming checkpoint-436 is the correct one for NL answers, adjust if needed

finetuned_model_nl_loaded = load_finetuned_model(base_model_id, adapter_path_nl)
print("Fine-tuned NL model loaded successfully!")


response_cache_path_nl = "finetuned_nl_responses_from_checkpoint"
df_response_nl_finetuned = eval_fine_tune(finetuned_model_nl_loaded, response_cache_p
ath_nl, split="val", prompt_type="standard_nl", response_type="nl")

print("\nEvaluation results for Fine-tuned NL Responses:")
evaluator_nl_finetuned = BaseEvaluator(df_response_nl_finetuned, truth_col="natural_l
ang_answers", pred_col="model_response")
evaluator_nl_finetuned.evaluate_exact_partial(id=False) # id=False for NL evaluation
### END TODO ###
```

```
Fine-tuned NL model loaded successfully!
Loading cached responses...
Cached responses loaded successfully!

Evaluation results for Fine-tuned NL Responses:


Metric          Exact           Partial
-----------------------------------------------
Precision       0.27            0.39
Recall          0.26            0.38
F1              0.25            0.36
```

Out[ ]:

	id	entities	natural_lang_entities	times	question	original_answers	natural_lang_answe
0	9709	['Q7414']	['The Walt Disney Company']	[]	what was disneys first color movie	['Q134430']	['Snow White and t Seven Dwar
1	9710	['Q717']	['Venezuela']	[1996]	at what rate was inflation in venezuela in the...	['+103.2']	['+103
2	9711	['Q17427', 'Q148']	['Communist Party of China', "People's Republi...	[]	which leader of the first communist party of c...	['Q369681']	['Chen Dux
3	9712	['Q4090718', 'Q4899742', 'Q4797909']	['Oliver Baldwin, 2nd Earl Baldwin of Bewdley'...	[]	who child of has brother of arthur baldwin, 3r...	['Q166635']	['Stanley Baldw
4	9713	['Q207191', 'Q191780']	['John C. Calhoun', 'ordinal number']	[]	what is position held of john c. calhoun that ...	['Q14213']	['United Stat Secretary of Stat
...	...	...	...	...	...	...	
3231	12940	['Q34201']	['Zeus']	[]	who is zeus oldest sister	['Q41419']	['Hest
3232	12941	['Q369706', 'Q182870']	['47th Academy Awards', 'Mario Puzo']	[]	what is award received from mario puzo which i...	['Q107258']	['Academy Award Best Writing, Adapt Scre
3233	12942	['Q1167169', 'Q25999', 'Q1381762', 'Q5223442']	['The Arrangement', 'sail', 'Teenage Mutant Ni...	[]	who are the characters within the moment seaso...	['Q163792', 'Q1428628', 'Q312905', 'Q739652']	['Captain Flint', 'Lo John Silver', 'Calico
3234	12943	['Q9617']	['Arsenal F.C.']	[]	when did the arsenal won the event championshi...	['Q4583688', 'Q47510372', 'Q4588534', 'Q475105...	['1987 Football Leag Cup Final', '1937– Fo
3235	12944	['Q142']	['France']	[2012, 2007]	what french leader in 2012 started their tenur...	['Q157', 'Q329']	['François Hollanc 'Nicolas Sarkoz

3236 rows × 24 columns

In [ ]:

```
base_model_id = "meta-llama/Llama-3.2-3B-Instruct"
adapter_path_id = f"{DATA_PATH}/cache/qlora-original_answers/checkpoint-436" # Assumi
ng checkpoint-436 is the correct one for ID answers, adjust if needed

finetuned_model_id_loaded = load_finetuned_model(base_model_id, adapter_path_id)
print("Fine-tuned ID model loaded successfully!")


response_cache_path_id = "finetuned_id_responses_from_checkpoint"
df_response_id_finetuned = eval_fine_tune(finetuned_model_id_loaded, response_cache_p
ath_id, split="val", prompt_type="standard_id", response_type="id")

print("\nEvaluation results for Fine-tuned ID Responses:")
evaluator_id_finetuned = BaseEvaluator(df_response_id_finetuned, truth_col="original_
answers", pred_col="model_response")
evaluator_id_finetuned.evaluate_exact_partial(id=True)
```

```
Device set to use cuda:0

Fine-tuned ID model loaded successfully!
Generating prompts for val dataset...
Creating prompts for 3236 questions...
Created 3236 prompts
Generating responses for val dataset...
Generating responses for 3236 prompts in batches of 16

You seem to be using the pipelines sequentially on GPU. In order to maximize efficie
ncy please use a dataset

Generated 3236 responses
Generated 3236 responses
Saved responses to /content/drive/MyDrive/NLP/TP4_Amaury/data/cache/finetuned_id_res
ponses_from_checkpoint.csv

Evaluation results for Fine-tuned ID Responses:


Metric          Exact
----------------------------------------------
Precision       0.03
Recall          0.03
F1              0.03
```

Out[ ]:

	id	entities	natural_lang_entities	times	question	original_answers	natural_lang_answe
0	9709	['Q7414']	['The Walt Disney Company']	[]	what was disneys first color movie	['Q134430']	['Snow White and t Seven Dwar
1	9710	['Q717']	['Venezuela']	[1996]	at what rate was inflation in venezuela in the...	['+103.2']	['+103
2	9711	['Q17427', 'Q148']	['Communist Party of China', "People's Republi...	[]	which leader of the first communist party of c...	['Q369681']	['Chen Dux
3	9712	['Q4090718', 'Q4899742', 'Q4797909']	['Oliver Baldwin, 2nd Earl Baldwin of Bewdley'...	[]	who child of has brother of arthur baldwin, 3r...	['Q166635']	['Stanley Baldw
4	9713	['Q207191', 'Q191780']	['John C. Calhoun', 'ordinal number']	[]	what is position held of john c. calhoun that ...	['Q14213']	['United Stat Secretary of Stat
...	...	...	...	...	...	...	
3231	12940	['Q34201']	['Zeus']	[]	who is zeus oldest sister	['Q41419']	['Hest
3232	12941	['Q369706', 'Q182870']	['47th Academy Awards', 'Mario Puzo']	[]	what is award received from mario puzo which i...	['Q107258']	['Academy Award Best Writing, Adapt Scre
3233	12942	['Q1167169', 'Q25999', 'Q1381762', 'Q5223442']	['The Arrangement', 'sail', 'Teenage Mutant Ni...	[]	who are the characters within the moment seaso...	['Q163792', 'Q1428628', 'Q312905', 'Q739652']	['Captain Flint', 'Lo John Silver', 'Calico
3234	12943	['Q9617']	['Arsenal F.C.']	[]	when did the arsenal won the event championshi...	['Q4583688', 'Q47510372', 'Q4588534', 'Q475105...	['1987 Football Leag Cup Final', '1937– Fc
3235	12944	['Q142']	['France']	[2012, 2007]	what french leader in 2012 started their tenur...	['Q157', 'Q329']	['François Hollanc 'Nicolas Sarkoz

3236 rows × 24 columns

Q 5.2.2 (2 pts) Observez et expliquez la différence de performance entre le modèle initial et le modèle affiné.

Modèle Initial

Résultats obtenus suite aux réponses sous forme de langage naturel:

	$Exact$	$Partial$
$Precision$	0.12	0.34
$Recall$	0.10	0.30
$F1$	0.12	0.30

Résultats obtenus suite aux réponses sous forme d'ID:

	$Exact$
$Precision$	0.0
$Recall$	0.0
$F1$	0.0

Modèle Fine-Tuned

Résultats obtenus suite aux réponses sous forme de langage naturel:

	$Exact$	$Partial$
$Precision$	0.27	0.39
$Recall$	0.26	0.28
$F1$	0.25	0.36

Résultats obtenus suite aux réponses sous forme d'ID:

	$Exact$
$Precision$	0.03
$Recall$	0.03
$F1$	0.03

Après l’affinage du modèle LLaMA-3B Instruct avec Q-LoRA sur notre dataset, nous pouvons observer une amélioration des diffférentes métriques pour les scores sous forme de langage naturel et sous forme d'ID. En effet, l'ensemble des métriques ont subit une lègere augmentation grâce à l'utilisation du modèle Fine-Tuned.

Le fine tunning a pour objectif de spécialiser un modèle pré-entraîné. En effectuant cette étape, nous permettons au modèle de langue, LLaMA-3B Instruct dans notre cas, d'ajuster ces poids pour s'adapter précisement à la tâche demandée. Cette spécialisation permet ainsi au modèle de fournir des réponses plus précise participant ainsi à une amélioration des scores. Ainsi, la différence de performance entre le modèle initial et le modèle affiné s'explique par l'affinage du modèle, permettant au modèle de générer des réponses avec une plus grande précision.

Partie 6 : Compétition Kaggle (40 pts)

Le reste de du TP est une compétition Kaggle dont voici le lien : https://www.kaggle.com/t/535bfa42807c4222bd3be5e72e410577 (https://www.kaggle.com/t/535bfa42807c4222bd3be5e72e410577).

Le but est de développer une nouvelle stratégie pour obtenir le meilleur score sur les IDs. Vous pouvez soumettre des prédictions sur l'ensemble de test (sous format `model_response.csv` , comme dans plus haut dans le TP) sur la compétition, et un score final sur un sous-ensemble caché de l'ensemble de test sera obtenu à la fin de la compétition.

Veillez à ce que votre solution soit exécutable en une seule connection Google Colab avec GPU (environ 3h maximum). Aussi, vous devez vous limiter aux modèle utilisés dans les parties précédentes du TP, soit Llama-3.2-3B-Instruct (https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct) et all-MiniLM-L6-v2 (https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2). Ceci assure que la compétition ne sera pas influencée par la capacité de calcul dont vous disposez.

Pour rappel, vous ne pouvez pas utiliser de données externes pour entraîner vos modèles, seulement les données fournies, et vous devez utiliser le même modèle de génération que celui utilisé dans les parties précédentes.

Proposez une architecture plus complexe, ou améliorez les modèles des parties précédentes, afin d'obtenir un score plus élevé que vos autres modèles sur l'ensemble de validation ET dans la compétition Kaggle. Démontrez bien cette amélioration dans votre notebook dans les cellules suivantes. Un autre objectif est d'obtenir le meilleur score de la classe.

Vous devez commencer cette démarche par un état de l'art.

6.1. État de l'art (5 points)

Décrivez en **deux paragraphes**, dans une cellule du notebook, avec les références appropriées, les approches de l'état de l'art pour la tâche que vous tentez d'accomplir. Utilisez une recherche dans Google Scholar.

Synthèse:

Les systèmes de Retrieval-Augmented Generation (RAG) représentent aujourd'hui l'état de l'art pour les tâches de question-réponse nécessitant des informations précises et factuelles. Contrairement aux modèles de langage purement paramétriques, les systèmes RAG consultent une base externe pour récupérer des passages pertinents avant la génération. Cela réduit considérablement les hallucinations et permet de mieux traiter les questions nécessitant des faits exacts. Le modèle REALM (Guu et al., 2020) et l'approche RAG de Lewis et al. (2020) ont montré que l'ajout d'un module de récupération dense améliore significativement la précision des réponses dans les tâches open-book QA et knowledge-intensive NLP. Ces travaux démontrent que la combinaison embeddings + large language models est aujourd'hui une stratégie centrale dans le domaine.

Des travaux plus récents se concentrent sur la spécialisation du retrieval, l'ajout de ré-ranking, et l'adaptation au domaine. Siriwardhana et al. (2023) ont montré qu'adapter le recovery aux données spécifiques améliore fortement les performances. Enfin, l'usage du fine-tuning léger, tel que LoRA, est désormais une technique dominante pour rendre un générateur plus performant sans coût d'entraînement prohibitif. Ces tendances indiquent que les meilleures architectures actuelles combinent Retrieval dense + Fine-Tuning + Prompt Engineering ciblé.

Synthèse: Le meilleur compromis performance / coût aujourd'hui : RAG (retriever dense) + générateur LLM fine-tuné + prompts enrichis.

Références:

[1] Lewis et al. — Retrieval-Augmented Generation for Knowledge-Intensive NLP (NeurIPS 2020) https://arxiv.org/abs/2005.11401 (https://arxiv.org/abs/2005.11401)

[2] Guu et al. — REALM: Retrieval-Augmented Language Model Pre-Training (ICML 2020) https://arxiv.org/abs/2002.08909 (https://arxiv.org/abs/2002.08909)

[3] Siriwardhana et al. — Improving Domain Adaptation of Retrieval-Augmented Generation (TACL 2023) https://doi.org/10.1162/tacl_a_00530 (https://doi.org/10.1162/tacl_a_00530)

[4] Hu et al. — LoRA: Low-Rank Adaptation for LLMs (ICLR 2022) https://arxiv.org/abs/2106.09685 (https://arxiv.org/abs/2106.09685)

In [ ]:
```
#@title Exemple de soumission
sample_submission = pd.read_csv(f"{DATA_PATH}/sample_submission.csv")
sample_submission.head()
```

6.2. Description de votre méthode (5 points)

Décrivez en deux paragraphes l'architecture proposée en soulignant notamment les additions ou l'originalité de l'approche et comment l'état de l'art l'a inspirée

Notre architecture suit les principes de l’état de l’art afin d’améliorer la qualité des prédictions dans la compétition Kaggle. Nous utiliserons tout d’abord un retriever dense basé sur des embeddings pour sélectionner les Top-8 passages les plus pertinents pour chaque question. Ensuite, nous construirons un prompt enrichi qui combine la question d'origine avec les passages récupérés. Enfin, la génération des données de la compétition sera effectuée par Llama-3.2-3B-Instruct fine-tuné avec QLoRA afin d'améliorer la capacité du modèle à prédire des identifiants corrects plutôt qu’un texte libre.

Nous intégrerons également deux techniques inspirées de l’état de l’art :

- Retrieval-Conditioned Prompting: L'objectif de cette méthode consiste à injecter directement les faits dans les prompts afin d’ancrer la prédiction dans un contexte fiable.
- Post-traitement structuré: Cette fonction extraira de manière robuste le QID généré par le modèle afin de garantir un format cohérent et d'éviter toute reformulation inutile.

  Cette architecture collaborative (Retrieval + LLM Fine-tuné + Prompt Engineering) devrait permettre d'amélioree les performances tant sur l’ensemble de validation que sur la plateforme Kaggle, tout en respectant les contraintes d’inférence rapide.

6.3. Implémentation (20 points)

Implémentez votre modèle ci-dessous dans une ou plusieurs cellules (veillez à la qualité de votre code) et effectuez votre soumission sur Kaggle en respectant le format de soumission.

In [ ]:

```
ADAPTER_PATH_ID = f"{DATA_PATH}/cache/qlora-original_answers"
BASE_MODEL_ID = "meta-llama/Llama-3.2-3B-Instruct"

# CHARGEMENT DU MODÈLE AFFINÉ
print(f"Chargement du modèle affiné (ID) depuis {ADAPTER_PATH_ID}...")
# On utilise la fonction load_finetuned_model définie en Partie 5
ft_model_kaggle = load_finetuned_model(BASE_MODEL_ID, ADAPTER_PATH_ID)

# PIPELINE ET RETRIEVER
kaggle_pipe = pipeline(
    "text-generation",
    model=ft_model_kaggle,
    tokenizer=tokenizer,
    dtype=torch.bfloat16,
    device_map="auto",
)

# On s'assure que le retriever est prêt (réutilisation de la Partie 4)
if 'retriever' not in locals():
    print("Initialisation du Retriever...")
    documents_list = facts_dataframe['fact'].to_list()
    retriever = CustomRetriever(
        documents_list,
        cache_file=f"{DATA_PATH}/cache/embeddings_cache.pkl",
        batch_size=16
    )

# FONCTION DE PRÉDICTION KAGGLE
def generate_kaggle_predictions(dataset_split, retriever, pipe, prompt_manager):
    """
    Pipeline complet : Retrieval -> Prompt (avec faits) -> Génération (Modèle Affiné)
    """
    questions = dataset_split['question'].to_list()
    ids = dataset_split['id'].to_list()

    #Retrieval (Top-K = 5 pour avoir assez de contexte)
    print(f"1. Récupération des faits pour {len(questions)} questions...")
    _, retrieved_texts = batch_retrieve(
        questions,
        retriever,
        batch_size=32,
        top_k=8
    )

    # Création des Prompts
    print("2. Construction des prompts RAG...")
    # On utilise 'standard_id' car notre modèle a été entraîné sur ce format
    base_prompts = create_all_prompts(
        dataset_df=dataset_split,
        prompt_manager=prompt_manager,
        prompt_type="standard_id"
    )

    # Injection des faits dans les prompts
    rag_prompts = []
    for i, prompt_msgs in enumerate(base_prompts):
        # Ajout des faits au dernier message utilisateur
        msg_with_facts = add_facts_to_prompt(prompt_manager, prompt_msgs, retrieved_t
exts[i])
        rag_prompts.append(msg_with_facts)

    # Génération
    print("3. Génération des réponses...")
```

```
    generation_config = {
        'max_new_tokens': 10, # Court car on attend des IDs
        'temperature': 0.001,   # Faible température pour la précision
        'top_p': 0.9,
        'batch_size': 16
    }

    responses = generate_all_responses_batch(
        pipe=pipe,
        prompts=rag_prompts,
        **generation_config
    )

    # Post-traitement (Nettoyage)
    cleaned_responses = []
    for resp in responses:
        # On prend tout ce qui ressemble à un QID au début
        clean = resp.strip().split()[0] if resp.strip() else "none"
        # Fallback simple
        if not clean.startswith("Q") and clean.lower() != "none":
             # Si le modèle bavarde encore, on essaie de trouver un QID
             import re
             match = re.search(r'Q\d+', resp)
             if match:
                 clean = match.group(0)
        cleaned_responses.append(clean)

    return ids, cleaned_responses

# EXÉCUTION SUR LE TEST SET
print(">>> Lancement de la génération pour la soumission Kaggle (Test Set)...")
test_ids, test_preds = generate_kaggle_predictions(dataset['test'], retriever, kaggle
_pipe, pm)

# CRÉATION DU FICHIER DE SOUMISSION
submission_df = pd.DataFrame({
    'id': test_ids,
    'model_response': test_preds
})

output_file = "submission_rag_ft.csv"
submission_df.to_csv(output_file, index=False)
print(f"\nFichier de soumission généré : {output_file}")
display(submission_df.head())
```

```
Chargement du modèle affiné (ID) depuis /content/drive/MyDrive/TP4_Aziz/data/cache/qlora-original_answers...
Loading checkpoint shards: 100% 2/2 [00:06<00:00,  2.94s/it]
Device set to use cuda:0
>>> Lancement de la génération pour la soumission Kaggle (Test Set)...
1. Récupération des faits pour 3237 questions...
Computing embeddings for 3237 queries in batches...
Batch Retrieval: 100%|| 102/102 [01:33<00:00,  1.09it/s]
2. Construction des prompts RAG...
Creating prompts for 3237 questions...
Created 3237 prompts
3. Génération des réponses...
Generating responses for 3237 prompts in batches of 16
You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset
Generated 3237 responses

Fichier de soumission généré : /content/drive/MyDrive/TP4_Aziz/data/submission_rag_ft.csv
```

	id	model_response
0	12945	['1994']
1	12946	['Nobel
2	12947	['The
3	12948	Q1340
4	12949	['1972']

In [ ]:
```python
# 6.4.1 ÉVALUATION SUR LE VALIDATION SET
print(">>> Évaluation de la méthode sur le Validation Set...")

#Générer les prédictions sur VAL
val_ids, val_preds = generate_kaggle_predictions(dataset['val'], retriever, kaggle_pi
pe, pm)

#Créer le DataFrame de résultats pour l'évaluateur
df_val_results = dataset['val'].copy()
df_val_results['model_response'] = val_preds

#Évaluation
evaluator_kaggle = BaseEvaluator(
    df_val_results,
    truth_col="original_answers",  # On compare avec les IDs réels
    pred_col="model_response"
)

scores_kaggle = evaluator_kaggle.evaluate_exact_partial(id=True)

print("\n=== RÉSULTATS DE L'ARCHITECTURE FINALE (RAG + FT) ===")
print(f"Exact Match F1 : {scores_kaggle['f1_exact'].mean():.4f}")
print(f"Partial Match F1: {scores_kaggle['f1_partial'].mean():.4f}")

# --- 6.4.2 COMPARAISON ---
print("\n--- Comparaison des performances (F1 Exact) ---")
print(f"1. Zero-shot (Base) :   [Voir Q3.5]")
print(f"2. RAG seul (Base) :    [Voir Q4.2]")
print(f"3. Fine-Tuning seul :   [Voir Q5.2]")
print(f"4. RAG + Fine-Tuning :  {scores_kaggle['f1_exact'].mean():.4f} (Notre méthod
e)")
```

```
>>> Évaluation de la méthode sur le Validation Set...
1. Récupération des faits pour 3236 questions...
Computing embeddings for 3236 queries in batches...
Batch Retrieval: 100%|██████████| 102/102 [01:33<00:00,  1.09it/s]
2. Construction des prompts RAG...
Creating prompts for 3236 questions...
Created 3236 prompts
3. Génération des réponses...
Generating responses for 3236 prompts in batches of 16
Generated 3236 responses


Metric          Exact
----------------------------------------------
Precision       0.18
Recall          0.16
F1              0.16

=== RÉSULTATS DE L'ARCHITECTURE FINALE (RAG + FT) ===
Exact Match F1 : 0.1623
Partial Match F1: 0.1635

--- Comparaison des performances (F1 Exact) ---
1. Zero-shot (Base) :   [Voir Q3.5]
2. RAG seul (Base) :    [Voir Q4.2]
3. Fine-Tuning seul :   [Voir Q5.2]
4. RAG + Fine-Tuning :  0.1623 (Notre méthode)
```

In [ ]:

```
import pandas as pd
import ast
import re

# 1 : Charger le sample Kaggle pour avoir les bons IDs + bon ordre
sample = pd.read_csv(f"{DATA_PATH}/sample_submission.csv")
print("Taille sample Kaggle : ", sample.shape)

# 2: Lire l'ancien fichier de soumission en brut
with open(f"{DATA_PATH}/submission_rag_ft.csv", encoding="utf-8") as f:
  lines = f.read().splitlines()

print("Nb lignes fichier (avec header) :", len(lines))

# On enleve la premiere ligne (header)
raw_lines = lines[1:]

raw_preds = []
for line in raw_lines:
  line = line.strip()
  if not line:
    continue

  # a) prendre tout ce qui vient apres la premiere virgule :
  if ',' in line:
    rest = line.split(',', 1)[1].strip()
  else:
    rest = line

  # b) garder seulement la vraie "liste" a partir du dernier '['
  if '[' in rest:
    rest = rest[rest.rfind('['):]
  raw_preds.append(rest)

print("Nb de predictions brutes recuperees : ", len(raw_preds))

# Ajuster la longueur au sample (truncate/pad)
if len(raw_preds) > len(sample):
  raw_preds = raw_preds[:len(sample)]
elif len(raw_preds) < len(sample):
  raw_preds += ["['Q000000']"] * (len(sample) - len(raw_preds))

def normalize_pred(s):
  s = str(s).strip()

  # 1 Esssayer de parser comme liste Python
  if s.startswith('[') and s.endswith(']'):
    try:
      lst = ast.literal_eval(s)
      if isinstance(lst, (list, tuple)) and len(lst) > 0:
        cleaned = [
            str(x).strip().strip('"').strip("'")
            for x in lst if str(x).strip()
        ]

        if cleaned:
          return repr(cleaned)
    except Exception:
      pass

  # 2 Si parsing KO: recuperer les QIDs + nombres
  ids = []
  qids = re.findall(r"Q\d+", s)
```

```
  ids.extend(qids)

  nums = re.findall(r"\d{4}|\d+", s)
  for n in nums:
    if n not in ids:
      ids.append(n)

  if ids:
    return repr(ids)

  # 3 Vraiment rien d'exploitable
  return "['Q000000']"

clean_preds = [normalize_pred(p) for p in raw_preds]

# 3 Construire un CSV propre pour Kaggle
submission_fixed = pd.DataFrame({
    "id": sample["id"],
    "prediction": clean_preds,
})

print(submission_fixed.head(10))
print("NaN : ", submission_fixed.isna().sum())
print("Shape final : ", submission_fixed.shape)

# 4 sauvegarder
output_file = f"{DATA_PATH}/submission_rag_ft_fixed.csv"
submission_fixed.to_csv(output_file, index=False)
print(f"\nFichier pret pour Kaggle : {output_file}")
```

```
Taille sample Kaggle :  (3237, 2)
Nb lignes fichier (avec header) : 3238
Nb de predictions brutes recuperees :  3237
      id         prediction
0  12945           ['1994']
1  12946        ['Q000000']
2  12947        ['Q000000']
3  12948  ['Q1340', '1340']
4  12949           ['1972']
5  12950        ['Q000000']
6  12951        ['Q000000']
7  12952           ['1998']
8  12953  ['Q1340', '1340']
9  12954        ['Q000000']
NaN :  id            0
prediction    0
dtype: int64
Shape final :  (3237, 2)

Fichier pret pour Kaggle : /content/drive/MyDrive/TP4_Aziz/data/submission_rag_ft_fixed.csv
```

In [ ]:
```
sample_submission = pd.read_csv(f"{DATA_PATH}/submission_rag_ft_fixed.csv")
sample_submission.head()
```

	id	prediction
0	12945	['1994']
1	12946	['Q000000']
2	12947	['Q000000']
3	12948	['Q1340', '1340']
4	12949	['1972']

6.4. Évaluation (4 points)

6.4.1. Évaluez votre modèle sur le jeu de validation en utilisant la métrique de F1 sur le Exact Match (2 points)

6.4.2 Comparez les résultats obtenus avec les modèles précédents. (2 points)

Résultats obtenus suite aux réponses sous forme d'ID sur le jeu de validation:

	$Exact$
$Precision$	0.18
$Recall$	0.16
$F1$	0.16

En combinant un module de récupération RAG avec un fine-tuning supervisé, nous avons réussi à atteindre un score F1 de 0.16.

Ainsi, nous pouvons remarquer une hausse significatives des résultats par rapport aux précédents modéles. En effet, avec un score F1 de 0.16 sur le jeu de validation, cette nouvelle architecture dépasse amplement les scores obtenus sur les identifiants Wikidata avec les précédents modèles étudiés lors de ce TP.

Cette augmentation significative des résultats s'explique notamment par cette nouvelle implémentation qui s'inpire de l'état de l'art et de plusieurs aspects du TP afin d'améliorer la prédictions des identifiants Wikidata.

6.5. Analyse (6 points)

6.5.1. Avantage/limites/Erreurs types (4 points)

En quelques phrases précises, discutez des avantages et limites de votre meilleure architecture. Analysez les cas d’erreur.

6.5.2. Améliorations potentielles (2 points)

Indiquez deux pistes d’amélioration futures potentielles et pourquoi vous pensez qu'elles permettraient d'obtenir de meilleurs résultats en vous basant sur un raisonnement logique ou sur des références à l'état de l'art

Avantages

Dans un premier temps, l’injection directe des faits dans le prompt permet de construire un prompt enrichi à partir des passages les plus pertinents sélectionnés par le retriever. Cette technique permet de guider le modèle vers une génération efficace des Q-ID en réduisant notamment le risque d’hallucination. Ensuite, comme étudié dans la partie 5, l'utilisation d’un modèle LLM ajusté pour la tâche de prédiction des identifiants Wikidata permet d'améliorer la qualité la précision des réponses. Enfin, le post-traitement robuste permet d'éliminer l'ensemble des tokens superflus assurant ainsi l'extraction et la production d’un Q-ID conforme au format attendu.

Limites

Malgré ces avantages, l’architecture présente plusieurs limites. Dans un premier, elle dépend de la qualité du retriever. Si le retriever ne sélectionne que des passages peu pertinents parmi les Top-8, le modèle de génération peut avoir des difficulté à produire un Q-ID correct. De plus, la génération des identifiants Q-ID est limité par les capacités du modèle. En effet, le modèle utilisé peut manquer de capacité pour comprendre les relations entre la question, le prompt et les faits retrouvés, diminuant par conséquent la précision du modèle.

Erreurs Types

Dans un premier temps, nous avons oberservés que certaines réponses n'étaient transformer en Q-ID. Cette oubli de transformation peut notamment s'expliquer par la longueur du prompt. En effet, l'ajout des faits dans le prompt peut nuire aux performances du modèle en lui faisant oublier la tâche principale à savoir la transformation en identifiants Q-ID.

De plus, nous pouvons remarquer via les résultats de la soumission Kaggle que de nombreuses réponses sont sous la forme Q00000. Cette forme indique que le modèle a retourné une réponse ne correspondant pas aux formats des Q-ID. Ainsi, cette erreur démontre une certaine incapacité du modèle à transformer les réponses sous forme de Q-ID corrects.

Améliorations potentielles

Afin d’améliorer la précision de notre modèle, nous pourrions envisager plusieurs possibilités. Tout d’abord, l’ajout d’exemples de réponses via la méthode few-shot permettrai de guider le modèle en lui fournissant des contextes et des formats de réponses attendus. Ces exemples pourraient notamment favoriser des résultats plus pertinents en évitant le problème des réponses Q0000 évoquée dans les erreurs types. Ensuite, nous pourrions réduire les paramètres de top_p afin de favoriser des sorties plus déterministes. Cette réduction pourrait limiter la variabilité des réponses et ainsi limiter les sorties inadéquates. Puis, une légère augmentation du nombre de documents récupérés par le retriever pourrait favoriser la multiplication d'information pertinentes afin de réduire le risque de générer des réponses incompletes ou incorrectes. Cependant, ce réglage doit être utilisé avec prudence, car un contexte trop long pourrait induire le modèle en erreur et dégrader sa précision.Enfin, nous pourrions ajouter un module de validation pour vérifier l'existence des Q-ID.