Explication cellule par cellule : detection\_intention.ipynb

Fichier source: notebooks/detection\_intention.ipynb

## Cellule 1 - MARKDOWN

Contenu (Markdown) :

# Detection d'Intention avec CamemBERT (Transformers)

\*\*Notebook optimise pour Google Colab GPU avec Transformers\*\*

Ce notebook utilise CamemBERT pour :

1. Classification d'intent (TRIP, NOT\_TRIP, UNKNOWN, NOT\_FRENCH)

2. Named Entity Recognition (Departure, Destination)

\*\*Ameliorations par rapport au baseline TF-IDF\*\* :

- Comprehension semantique profonde du texte

- Detection precise des langues etrangeres

- NER contextuel (pas juste un gazetteer)

- Fine-tuning sur donnees francaises

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 2 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: # Verification GPU et installation des dependances

--> Explication : Commentaire expliquant l'intention du bloc de code.

002: import torch

--> Explication : Import de module(s) : torch. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

003:

--> Explication : Ligne vide (séparateur / lisibilité).

004: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

005: print('DETECTION INTENTION - Transformers + CamemBERT')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

006: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

007:

--> Explication : Ligne vide (séparateur / lisibilité).

008: print(f'\nPyTorch version: {torch.\_\_version\_\_}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

009: print(f'CUDA disponible: {torch.cuda.is\_available()}')

--> Explication : Vérifie la disponibilité d'un GPU CUDA pour accélérer l'entraînement. Si disponible, adapte la configuration (fp16, device).

010:

--> Explication : Ligne vide (séparateur / lisibilité).

011: if torch.cuda.is\_available():

--> Explication : Vérifie la disponibilité d'un GPU CUDA pour accélérer l'entraînement. Si disponible, adapte la configuration (fp16, device).

012: print(f'GPU: {torch.cuda.get\_device\_name(0)}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

013: print(f'Memoire GPU: {torch.cuda.get\_device\_properties(0).total\_memory / 1e9:.2f} GB')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

014: else:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

015: print('ATTENTION: Pas de GPU detecte!')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

016: print('Allez dans Runtime > Change runtime type > GPU')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

017:

--> Explication : Ligne vide (séparateur / lisibilité).

018: # Installation des packages Transformers

--> Explication : Commentaire expliquant l'intention du bloc de code.

019: !pip install -q transformers datasets evaluate seqeval accelerate scikit-learn langdetect

--> Explication : Détection automatique de la langue du texte pour classer NOT\_FRENCH le cas échéant.

020: print('\nInstallation terminee!')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- CamemBERT est choisi pour ses performances sur le français (pré-entraîné sur OSCAR FR).

- Seqeval est utilisé pour mesurer la qualité du NER (precision/recall/F1 par entité).

- Utilisation de fp16 et GPU si disponible pour accélérer l'entrainement et réduire mémoire.

## Cellule 3 - MARKDOWN

Contenu (Markdown) :

## Montage Google Drive et desactivation WandB

Monte Google Drive pour acceder aux datasets et desactive WandB (tracking non necessaire).

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 4 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from google.colab import drive

--> Explication : Import ciblé : from google.colab import drive. Permet d'importer des fonctions/classes spécifiques d'un package.

002: import os

--> Explication : Import de module(s) : os. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

003:

--> Explication : Ligne vide (séparateur / lisibilité).

004: # Monter Google Drive

--> Explication : Commentaire expliquant l'intention du bloc de code.

005: drive.mount('/content/drive')

--> Explication : Monte Google Drive pour accéder aux datasets et sauvegarder modèles depuis Colab.

006:

--> Explication : Ligne vide (séparateur / lisibilité).

007: # Desactiver WandB

--> Explication : Commentaire expliquant l'intention du bloc de code.

008: os.environ['WANDB\_DISABLED'] = 'true'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

009:

--> Explication : Ligne vide (séparateur / lisibilité).

010: # Chemins

--> Explication : Commentaire expliquant l'intention du bloc de code.

011: workdir = '/content/drive/MyDrive/dataset'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

012: os.makedirs(workdir, exist\_ok=True)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

013:

--> Explication : Ligne vide (séparateur / lisibilité).

014: print('Working directory:', workdir)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

015: print('WandB: DESACTIVE')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- Google Drive permet de stocker et partager datasets/modèles entre sessions Colab.

## Cellule 5 - MARKDOWN

Contenu (Markdown) :

## Chargement des datasets et analyse

Charge les datasets (train\_set.csv, test\_set.csv) et affiche les statistiques.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 6 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: import pandas as pd

--> Explication : Import de module(s) : pandas as pd. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

002:

--> Explication : Ligne vide (séparateur / lisibilité).

003: train\_path = os.path.join(workdir, 'train\_set.csv')

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

004: test\_path = os.path.join(workdir, 'test\_set.csv')

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

005:

--> Explication : Ligne vide (séparateur / lisibilité).

006: # Verification existence

--> Explication : Commentaire expliquant l'intention du bloc de code.

007: for p in [train\_path, test\_path]:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

008: if not os.path.exists(p):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

009: print(f'ERREUR: Fichier non trouve: {p}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

010: raise FileNotFoundError(p)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

011:

--> Explication : Ligne vide (séparateur / lisibilité).

012: # Chargement

--> Explication : Commentaire expliquant l'intention du bloc de code.

013: train\_df = pd.read\_csv(train\_path, encoding='utf-8')

--> Explication : Lecture d'un fichier CSV dans un DataFrame pandas.

014: test\_df = pd.read\_csv(test\_path, encoding='utf-8')

--> Explication : Lecture d'un fichier CSV dans un DataFrame pandas.

015:

--> Explication : Ligne vide (séparateur / lisibilité).

016: print(f'Train shape: {train\_df.shape}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

017: print(f'Test shape: {test\_df.shape}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

018:

--> Explication : Ligne vide (séparateur / lisibilité).

019: print('\nDistribution des classes (Train):')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

020: print(train\_df['intent'].value\_counts())

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

021: print(f'\nPourcentages:')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

022: print(train\_df['intent'].value\_counts(normalize=True).apply(lambda x: f'{x:.2%}'))

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

023:

--> Explication : Ligne vide (séparateur / lisibilité).

024: display(train\_df.head(3))

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

Choix et raisons (pour cette cellule) :

- Bloc de code utilitaire implémentant une étape standard (chargement, préprocessing, entraînement, évaluation).

## Cellule 7 - MARKDOWN

Contenu (Markdown) :

## Preprocessing et parsing des entities

Nettoie les donnees et parse les annotations JSON pour le NER.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 8 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: import json

--> Explication : Import de module(s) : json. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

002:

--> Explication : Ligne vide (séparateur / lisibilité).

003: # Nettoyer les colonnes necessaires

--> Explication : Commentaire expliquant l'intention du bloc de code.

004: train\_df = train\_df[['text', 'intent', 'entities']].dropna(subset=['text', 'intent']).reset\_index(drop=True)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

005: test\_df = test\_df[['text', 'intent', 'entities']].dropna(subset=['text', 'intent']).reset\_index(drop=True)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

006:

--> Explication : Ligne vide (séparateur / lisibilité).

007: def parse\_entities\_field(row):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

008: """Parse la colonne entities (JSON) et valide."""

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

009: try:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

010: ents = json.loads(row['entities']) if pd.notna(row['entities']) else []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

011: except:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

012: ents = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

013:

--> Explication : Ligne vide (séparateur / lisibilité).

014: valid = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

015: txt = row.get('text', '')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

016: for ent in ents:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

017: if isinstance(ent, dict) and 'start' in ent and 'end' in ent and 'label' in ent:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

018: if 0 <= ent['start'] < ent['end'] <= len(txt):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

019: valid.append(ent)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

020: return valid

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

021:

--> Explication : Ligne vide (séparateur / lisibilité).

022: # Parser les entities

--> Explication : Commentaire expliquant l'intention du bloc de code.

023: train\_df['parsed\_entities'] = train\_df.apply(parse\_entities\_field, axis=1)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

024: test\_df['parsed\_entities'] = test\_df.apply(parse\_entities\_field, axis=1)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

025:

--> Explication : Ligne vide (séparateur / lisibilité).

026: print('Entities parsees avec succes!')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

027: print(f'\nExemple TRIP avec entities:')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

028: trip\_ex = train\_df[train\_df['intent'] == 'TRIP'].iloc[0]

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

029: print(f' Texte: {trip\_ex["text"]}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

030: print(f' Entities: {trip\_ex["parsed\_entities"]}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- Bloc de code utilitaire implémentant une étape standard (chargement, préprocessing, entraînement, évaluation).

## Cellule 9 - MARKDOWN

Contenu (Markdown) :

## Preparation des datasets HuggingFace pour Intent Classification

Convertit les DataFrames en datasets HuggingFace et encode les labels.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 10 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from datasets import Dataset

--> Explication : Import ciblé : from datasets import Dataset. Permet d'importer des fonctions/classes spécifiques d'un package.

002: from sklearn.preprocessing import LabelEncoder

--> Explication : Import ciblé : from sklearn.preprocessing import LabelEncoder. Permet d'importer des fonctions/classes spécifiques d'un package.

003:

--> Explication : Ligne vide (séparateur / lisibilité).

004: # Encoder les labels d'intent

--> Explication : Commentaire expliquant l'intention du bloc de code.

005: label\_encoder = LabelEncoder()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

006: all\_intents = list(set(list(train\_df['intent'].unique()) + list(test\_df['intent'].unique())))

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

007: label\_encoder.fit(all\_intents)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

008:

--> Explication : Ligne vide (séparateur / lisibilité).

009: print('Classes d\'intent:')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

010: for i, label in enumerate(label\_encoder.classes\_):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

011: print(f' {i}: {label}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

012:

--> Explication : Ligne vide (séparateur / lisibilité).

013: # Creer les datasets HuggingFace

--> Explication : Commentaire expliquant l'intention du bloc de code.

014: train\_df['label'] = label\_encoder.transform(train\_df['intent'])

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

015: test\_df['label'] = label\_encoder.transform(test\_df['intent'])

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

016:

--> Explication : Ligne vide (séparateur / lisibilité).

017: intent\_train\_dataset = Dataset.from\_pandas(train\_df[['text', 'label']])

--> Explication : Conversion d'un DataFrame pandas en HuggingFace Dataset pour faciliter training/eval.

018: intent\_test\_dataset = Dataset.from\_pandas(test\_df[['text', 'label']])

--> Explication : Conversion d'un DataFrame pandas en HuggingFace Dataset pour faciliter training/eval.

019:

--> Explication : Ligne vide (séparateur / lisibilité).

020: print(f'\nIntent datasets crees:')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

021: print(f' Train: {len(intent\_train\_dataset)} exemples')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

022: print(f' Test: {len(intent\_test\_dataset)} exemples')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- Bloc de code utilitaire implémentant une étape standard (chargement, préprocessing, entraînement, évaluation).

## Cellule 11 - MARKDOWN

Contenu (Markdown) :

## Fine-Tuning Intent Classification avec CamemBERT

Entrainement du modele CamemBERT pour la classification d'intent.

\*\*Ameliorations clees\*\* :

- Comprehension semantique (vs TF-IDF)

- Gestion des langues etrangeres

- Transfer learning depuis modele pre-entraine francais

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 12 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from transformers import (

--> Explication : Import ciblé : from transformers import (. Permet d'importer des fonctions/classes spécifiques d'un package.

002: AutoTokenizer,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

003: AutoModelForSequenceClassification,

--> Explication : Chargement d'un tokenizer / modèle pré-entraîné depuis HuggingFace (CamemBERT), adapté au français.

004: TrainingArguments,

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

005: Trainer

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

006: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

007: from sklearn.utils.class\_weight import compute\_class\_weight

--> Explication : Import ciblé : from sklearn.utils.class\_weight import compute\_class\_weight. Permet d'importer des fonctions/classes spécifiques d'un package.

008: import torch.nn as nn

--> Explication : Import de module(s) : torch.nn as nn. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

009: import evaluate

--> Explication : Import de module(s) : evaluate. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

010: import numpy as np

--> Explication : Import de module(s) : numpy as np. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

011:

--> Explication : Ligne vide (séparateur / lisibilité).

012: # Tokenizer

--> Explication : Commentaire expliquant l'intention du bloc de code.

013: tokenizer = AutoTokenizer.from\_pretrained('camembert-base')

--> Explication : Chargement d'un tokenizer / modèle pré-entraîné depuis HuggingFace (CamemBERT), adapté au français.

014:

--> Explication : Ligne vide (séparateur / lisibilité).

015: def tokenize\_function(examples):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

016: return tokenizer(

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

017: examples['text'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

018: padding='max\_length',

--> Explication : Paramètres de tokenization pour limiter la longueur des séquences et éviter débordements.

019: truncation=True,

--> Explication : Paramètres de tokenization pour limiter la longueur des séquences et éviter débordements.

020: max\_length=128

--> Explication : Paramètres de tokenization pour limiter la longueur des séquences et éviter débordements.

021: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

022:

--> Explication : Ligne vide (séparateur / lisibilité).

023: print('Tokenization...')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

024: tokenized\_train = intent\_train\_dataset.map(tokenize\_function, batched=True)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

025: tokenized\_test = intent\_test\_dataset.map(tokenize\_function, batched=True)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

026: print('Tokenization terminee!')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

027:

--> Explication : Ligne vide (séparateur / lisibilité).

028: # Modele avec dropout augmente

--> Explication : Commentaire expliquant l'intention du bloc de code.

029: num\_labels = len(label\_encoder.classes\_)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

030: intent\_model = AutoModelForSequenceClassification.from\_pretrained(

--> Explication : Chargement d'un tokenizer / modèle pré-entraîné depuis HuggingFace (CamemBERT), adapté au français.

031: 'camembert-base',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

032: num\_labels=num\_labels,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

033: id2label={i: label for i, label in enumerate(label\_encoder.classes\_)},

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

034: label2id={label: i for i, label in enumerate(label\_encoder.classes\_)},

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

035: hidden\_dropout\_prob=0.4,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

036: attention\_probs\_dropout\_prob=0.4,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

037: classifier\_dropout=0.4

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

038: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

039:

--> Explication : Ligne vide (séparateur / lisibilité).

040: # Class weights

--> Explication : Commentaire expliquant l'intention du bloc de code.

041: class\_weights = compute\_class\_weight(

--> Explication : Calcul des poids de classes pour compenser le déséquilibre des classes durant l'entraînement.

042: class\_weight='balanced',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

043: classes=np.unique(train\_df['label']),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

044: y=train\_df['label']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

045: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

046:

--> Explication : Ligne vide (séparateur / lisibilité).

047: print('\nClass weights:')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

048: for label, weight in zip(label\_encoder.classes\_, class\_weights):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

049: print(f' {label}: {weight:.3f}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

050:

--> Explication : Ligne vide (séparateur / lisibilité).

051: # Trainer avec weighted loss (CORRIGE pour nouvelle API Transformers)

--> Explication : Commentaire expliquant l'intention du bloc de code.

052: class WeightedTrainer(Trainer):

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

053: def compute\_loss(self, model, inputs, return\_outputs=False, num\_items\_in\_batch=None):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

054: labels = inputs.get("labels")

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

055: outputs = model(\*\*inputs)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

056: logits = outputs.get("logits")

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

057:

--> Explication : Ligne vide (séparateur / lisibilité).

058: weight\_tensor = torch.tensor(class\_weights, dtype=torch.float).to(logits.device)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

059: loss\_fct = nn.CrossEntropyLoss(weight=weight\_tensor)

--> Explication : Fonction de perte / normalisation utilisée pour l'apprentissage des probabilités de classes.

060: loss = loss\_fct(logits.view(-1, self.model.config.num\_labels), labels.view(-1))

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

061:

--> Explication : Ligne vide (séparateur / lisibilité).

062: return (loss, outputs) if return\_outputs else loss

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

063:

--> Explication : Ligne vide (séparateur / lisibilité).

064: # Metriques

--> Explication : Commentaire expliquant l'intention du bloc de code.

065: accuracy\_metric = evaluate.load('accuracy')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

066: f1\_metric = evaluate.load('f1')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

067:

--> Explication : Ligne vide (séparateur / lisibilité).

068: def compute\_intent\_metrics(eval\_pred):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

069: logits, labels = eval\_pred

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

070: predictions = np.argmax(logits, axis=-1)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

071:

--> Explication : Ligne vide (séparateur / lisibilité).

072: from sklearn.metrics import f1\_score, precision\_score, recall\_score

--> Explication : Import ciblé : from sklearn.metrics import f1\_score, precision\_score, recall\_score. Permet d'importer des fonctions/classes spécifiques d'un package.

073:

--> Explication : Ligne vide (séparateur / lisibilité).

074: accuracy = (predictions == labels).mean()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

075: f1\_macro = f1\_score(labels, predictions, average='macro')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

076: f1\_per\_class = f1\_score(labels, predictions, average=None)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

077:

--> Explication : Ligne vide (séparateur / lisibilité).

078: metrics = {

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

079: 'accuracy': float(accuracy),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

080: 'f1\_macro': float(f1\_macro)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

081: }

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

082:

--> Explication : Ligne vide (séparateur / lisibilité).

083: for i, label in enumerate(label\_encoder.classes\_):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

084: metrics[f'f1\_{label}'] = float(f1\_per\_class[i])

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

085:

--> Explication : Ligne vide (séparateur / lisibilité).

086: return metrics

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

087:

--> Explication : Ligne vide (séparateur / lisibilité).

088: # Training arguments optimises

--> Explication : Commentaire expliquant l'intention du bloc de code.

089: training\_args = TrainingArguments(

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

090: output\_dir=os.path.join(workdir, 'models/intent\_classifier'),

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

091: num\_train\_epochs=6,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

092: per\_device\_train\_batch\_size=16,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

093: per\_device\_eval\_batch\_size=32,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

094: learning\_rate=8e-6,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

095: weight\_decay=0.03,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

096: eval\_strategy='epoch',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

097: save\_strategy='epoch',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

098: load\_best\_model\_at\_end=True,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

099: metric\_for\_best\_model='f1\_macro',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

100: logging\_steps=50,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

101: warmup\_steps=300,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

102: fp16=torch.cuda.is\_available(),

--> Explication : Vérifie la disponibilité d'un GPU CUDA pour accélérer l'entraînement. Si disponible, adapte la configuration (fp16, device).

103: push\_to\_hub=False,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

104: report\_to='none',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

105: gradient\_accumulation\_steps=2,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

106: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

107:

--> Explication : Ligne vide (séparateur / lisibilité).

108: # Trainer

--> Explication : Commentaire expliquant l'intention du bloc de code.

109: intent\_trainer = WeightedTrainer(

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

110: model=intent\_model,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

111: args=training\_args,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

112: train\_dataset=tokenized\_train,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

113: eval\_dataset=tokenized\_test,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

114: tokenizer=tokenizer,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

115: compute\_metrics=compute\_intent\_metrics

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

116: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

117:

--> Explication : Ligne vide (séparateur / lisibilité).

118: print('\n' + '='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

119: print('DEBUT DU FINE-TUNING AMELIORE')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

120: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

121:

--> Explication : Ligne vide (séparateur / lisibilité).

122: intent\_trainer.train()

--> Explication : Lancement de la boucle d'entraînement du modèle (fine-tuning).

123:

--> Explication : Ligne vide (séparateur / lisibilité).

124: print('\nFine-tuning termine!')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- CamemBERT est choisi pour ses performances sur le français (pré-entraîné sur OSCAR FR).

- Le dataset est déséquilibré ; on utilise des poids de classes pour éviter le biais vers la classe majoritaire.

- La tokenization avec padding/truncation assure des entrées de taille fixe pour le batching.

- TrainingArguments centralise la configuration d'entrainement (epoch, batch, lr) pour reproductibilité et réglage fin.

- Le NER en format BIO permet d'extraire des entités (Departure/Destination) même pour noms composés.

- Utilisation de fp16 et GPU si disponible pour accélérer l'entrainement et réduire mémoire.

## Cellule 13 - MARKDOWN

Contenu (Markdown) :

## Evaluation du modele Intent Classification

Evalue le modele sur le test set avec metriques detaillees et matrice de confusion.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 14 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from sklearn.metrics import classification\_report, confusion\_matrix

--> Explication : Import ciblé : from sklearn.metrics import classification\_report, confusion\_matrix. Permet d'importer des fonctions/classes spécifiques d'un package.

002: import matplotlib.pyplot as plt

--> Explication : Import de module(s) : matplotlib.pyplot as plt. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

003: import seaborn as sns

--> Explication : Import de module(s) : seaborn as sns. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

004:

--> Explication : Ligne vide (séparateur / lisibilité).

005: # Evaluer sur le test set

--> Explication : Commentaire expliquant l'intention du bloc de code.

006: eval\_results = intent\_trainer.evaluate()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

007:

--> Explication : Ligne vide (séparateur / lisibilité).

008: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

009: print('RESULTATS SUR LE TEST SET')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

010: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

011: for key, value in eval\_results.items():

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

012: if isinstance(value, float):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

013: print(f' {key}: {value:.4f}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

014:

--> Explication : Ligne vide (séparateur / lisibilité).

015: # Predictions detaillees

--> Explication : Commentaire expliquant l'intention du bloc de code.

016: predictions = intent\_trainer.predict(tokenized\_test)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

017: predicted\_labels = np.argmax(predictions.predictions, axis=-1)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

018: true\_labels = predictions.label\_ids

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

019:

--> Explication : Ligne vide (séparateur / lisibilité).

020:

--> Explication : Ligne vide (séparateur / lisibilité).

021: # Ajouter post-processing

--> Explication : Commentaire expliquant l'intention du bloc de code.

022: def post\_process\_prediction(text, predicted\_intent):

--> Explication : Fonction de post-traitement : règles heuristiques pour corriger des prédictions erronées.

023: """

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

024: Corrige les predictions aberrantes avec regles heuristiques.

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

025: """

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

026: text\_lower = text.lower().strip()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

027:

--> Explication : Ligne vide (séparateur / lisibilité).

028: # Detection langue

--> Explication : Commentaire expliquant l'intention du bloc de code.

029: try:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

030: from langdetect import detect

--> Explication : Import ciblé : from langdetect import detect. Permet d'importer des fonctions/classes spécifiques d'un package.

031: lang = detect(text) if len(text) >= 3 else 'unknown'

--> Explication : Détection automatique de la langue du texte pour classer NOT\_FRENCH le cas échéant.

032: except:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

033: lang = 'fr' # Par defaut francais

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

034:

--> Explication : Ligne vide (séparateur / lisibilité).

035: # REGLE 1: Langue non-francaise

--> Explication : Commentaire expliquant l'intention du bloc de code.

036: english\_markers = ['i ', 'you ', 'the ', 'is ', 'are ', 'have ', 'do ', 'can ', 'hello', 'please']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

037: spanish\_markers = ['el ', 'la ', 'los ', 'de ', 'que ', 'por ', 'para ', 'bueno', 'gracias']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

038: german\_markers = ['der ', 'die ', 'das ', 'ist ', 'ich ', 'sie ', 'haben ', 'guten', 'danke']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

039:

--> Explication : Ligne vide (séparateur / lisibilité).

040: if (lang not in ['fr', 'unknown']) or \

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

041: any(marker in text\_lower for marker in english\_markers + spanish\_markers + german\_markers):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

042: if predicted\_intent != 'NOT\_FRENCH':

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

043: return 'NOT\_FRENCH'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

044:

--> Explication : Ligne vide (séparateur / lisibilité).

045: # REGLE 2: Mots-cles NOT\_TRIP

--> Explication : Commentaire expliquant l'intention du bloc de code.

046: not\_trip\_keywords = [

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

047: 'merci', 'remercie', 'confirme', 'email', 'reunion', 'document',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

048: 'rapport', 'felicitation', 'bravo', 'compte-rendu', 'transferer'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

049: ]

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

050: if any(kw in text\_lower for kw in not\_trip\_keywords):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

051: trip\_keywords = ['billet', 'train', 'aller', 'retour', 'trajet']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

052: has\_trip = any(kw in text\_lower for kw in trip\_keywords)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

053: if not has\_trip and predicted\_intent == 'TRIP':

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

054: return 'NOT\_TRIP'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

055:

--> Explication : Ligne vide (séparateur / lisibilité).

056: # REGLE 3: Texte incomprehensible (tres court, pas de mots francais)

--> Explication : Commentaire expliquant l'intention du bloc de code.

057: if len(text\_lower) < 15 and lang == 'unknown':

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

058: words = text\_lower.split()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

059: if len(words) <= 3:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

060: # Verifier si ce sont des mots francais reconnaissables

--> Explication : Commentaire expliquant l'intention du bloc de code.

061: french\_common = ['de', 'a', 'le', 'la', 'pour', 'je', 'tu', 'il']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

062: if not any(word in french\_common for word in words):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

063: if predicted\_intent != 'UNKNOWN':

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

064: return 'UNKNOWN'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

065:

--> Explication : Ligne vide (séparateur / lisibilité).

066: # REGLE 4: Format "Ville1 Ville2" = TRIP

--> Explication : Commentaire expliquant l'intention du bloc de code.

067: words = text\_lower.split()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

068: if len(words) == 2 and len(text) < 50:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

069: # Verifier si ce sont deux noms propres (majuscules)

--> Explication : Commentaire expliquant l'intention du bloc de code.

070: if text.split()[0][0].isupper() and text.split()[1][0].isupper():

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

071: if predicted\_intent != 'TRIP':

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

072: return 'TRIP'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

073:

--> Explication : Ligne vide (séparateur / lisibilité).

074: # REGLE 5: Presence "de X a Y" ou "X Y" = TRIP

--> Explication : Commentaire expliquant l'intention du bloc de code.

075: if ('de ' in text\_lower and ' a ' in text\_lower) or \

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

076: ('de ' in text\_lower and ' vers ' in text\_lower) or \

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

077: (' pour ' in text\_lower and len(words) < 10):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

078: trip\_keywords = ['billet', 'train', 'horaire', 'tarif']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

079: not\_trip\_keywords\_strict = ['merci', 'document', 'rapport']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

080: has\_not\_trip = any(kw in text\_lower for kw in not\_trip\_keywords\_strict)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

081: if not has\_not\_trip and predicted\_intent != 'TRIP':

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

082: return 'TRIP'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

083:

--> Explication : Ligne vide (séparateur / lisibilité).

084: return predicted\_intent

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

085:

--> Explication : Ligne vide (séparateur / lisibilité).

086: # Appliquer post-processing

--> Explication : Commentaire expliquant l'intention du bloc de code.

087: test\_df\_eval = test\_df.copy()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

088: test\_df\_eval['predicted\_label'] = predicted\_labels

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

089: test\_df\_eval['predicted\_intent'] = label\_encoder.inverse\_transform(predicted\_labels)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

090:

--> Explication : Ligne vide (séparateur / lisibilité).

091: corrected\_intents = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

092: for idx, row in test\_df\_eval.iterrows():

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

093: corrected = post\_process\_prediction(row['text'], row['predicted\_intent'])

--> Explication : Fonction de post-traitement : règles heuristiques pour corriger des prédictions erronées.

094: corrected\_intents.append(corrected)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

095:

--> Explication : Ligne vide (séparateur / lisibilité).

096: test\_df\_eval['corrected\_intent'] = corrected\_intents

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

097: test\_df\_eval['corrected\_label'] = label\_encoder.transform(corrected\_intents)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

098:

--> Explication : Ligne vide (séparateur / lisibilité).

099: # Comparer

--> Explication : Commentaire expliquant l'intention du bloc de code.

100: accuracy\_before = (test\_df\_eval['label'] == test\_df\_eval['predicted\_label']).mean()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

101: accuracy\_after = (test\_df\_eval['label'] == test\_df\_eval['corrected\_label']).mean()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

102:

--> Explication : Ligne vide (séparateur / lisibilité).

103: print(f'\\nAccuracy avant post-processing: {accuracy\_before:.4f}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

104: print(f'Accuracy apres post-processing: {accuracy\_after:.4f}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

105: print(f'Amelioration: {(accuracy\_after - accuracy\_before):.4f} ({(accuracy\_after/accuracy\_before - 1)\*100:+.2f}%)')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

106:

--> Explication : Ligne vide (séparateur / lisibilité).

107: # Classification report apres PP

--> Explication : Commentaire expliquant l'intention du bloc de code.

108: from sklearn.metrics import classification\_report

--> Explication : Import ciblé : from sklearn.metrics import classification\_report. Permet d'importer des fonctions/classes spécifiques d'un package.

109: print('\\n' + '='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

110: print('CLASSIFICATION REPORT APRES POST-PROCESSING')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

111: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

112: print(classification\_report(

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

113: test\_df\_eval['label'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

114: test\_df\_eval['corrected\_label'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

115: target\_names=label\_encoder.classes\_,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

116: digits=4

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

117: ))

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

118:

--> Explication : Ligne vide (séparateur / lisibilité).

119: # Classification report

--> Explication : Commentaire expliquant l'intention du bloc de code.

120: print('\n' + '='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

121: print('CLASSIFICATION REPORT')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

122: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

123: print(classification\_report(

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

124: true\_labels,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

125: predicted\_labels,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

126: target\_names=label\_encoder.classes\_,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

127: digits=4

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

128: ))

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

129:

--> Explication : Ligne vide (séparateur / lisibilité).

130: # Matrice de confusion

--> Explication : Commentaire expliquant l'intention du bloc de code.

131: cm = confusion\_matrix(true\_labels, predicted\_labels)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

132: plt.figure(figsize=(10, 8))

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

133: sns.heatmap(

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

134: cm,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

135: annot=True,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

136: fmt='d',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

137: cmap='Blues',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

138: xticklabels=label\_encoder.classes\_,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

139: yticklabels=label\_encoder.classes\_

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

140: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

141: plt.title('Matrice de Confusion - Intent Classification')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

142: plt.ylabel('Vraie Classe')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

143: plt.xlabel('Classe Predite')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

144: plt.tight\_layout()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

145: plt.show()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

146:

--> Explication : Ligne vide (séparateur / lisibilité).

147: # Sauvegarder le modele

--> Explication : Commentaire expliquant l'intention du bloc de code.

148: model\_path = os.path.join(workdir, 'models/intent\_classifier\_best')

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

149: intent\_trainer.save\_model(model\_path)

--> Explication : Sauvegarde du modèle et tokenizer pour réutilisation en production/inference.

150: tokenizer.save\_pretrained(model\_path)

--> Explication : Sauvegarde du modèle et tokenizer pour réutilisation en production/inference.

151:

--> Explication : Ligne vide (séparateur / lisibilité).

152: print(f'\nModele sauvegarde dans: {model\_path}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- Les heuristiques post-traitement corrigent les erreurs systématiques du modèle (langue, mots-clés, formats ambigus).

- Le NER en format BIO permet d'extraire des entités (Departure/Destination) même pour noms composés.

## Cellule 15 - MARKDOWN

Contenu (Markdown) :

## Preparation du dataset NER (Token Classification)

Conversion des annotations en format BIO pour le NER contextuel avec Transformers.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 16 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from transformers import AutoTokenizer

--> Explication : Import ciblé : from transformers import AutoTokenizer. Permet d'importer des fonctions/classes spécifiques d'un package.

002:

--> Explication : Ligne vide (séparateur / lisibilité).

003: # Tokenizer rapide pour NER

--> Explication : Commentaire expliquant l'intention du bloc de code.

004: tokenizer\_fast = AutoTokenizer.from\_pretrained('camembert-base', use\_fast=True)

--> Explication : Chargement d'un tokenizer / modèle pré-entraîné depuis HuggingFace (CamemBERT), adapté au français.

005:

--> Explication : Ligne vide (séparateur / lisibilité).

006: # Labels NER en format BIO

--> Explication : Commentaire expliquant l'intention du bloc de code.

007: ner\_labels = ['O', 'B-Departure', 'I-Departure', 'B-Destination', 'I-Destination']

--> Explication : Conversion des annotations en format BIO pour l'entraînement du NER (Token Classification).

008: label2id = {label: i for i, label in enumerate(ner\_labels)}

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

009: id2label = {i: label for label, i in label2id.items()}

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

010:

--> Explication : Ligne vide (séparateur / lisibilité).

011: print('Labels NER:')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

012: for label\_id, label\_name in id2label.items():

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

013: print(f' {label\_id}: {label\_name}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

014:

--> Explication : Ligne vide (séparateur / lisibilité).

015: def convert\_to\_bio\_tags(text, entities):

--> Explication : Conversion des annotations en format BIO pour l'entraînement du NER (Token Classification).

016: """Convertit les annotations en tags BIO pour chaque token."""

--> Explication : Conversion des annotations en format BIO pour l'entraînement du NER (Token Classification).

017: encoding = tokenizer\_fast(

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

018: text,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

019: return\_offsets\_mapping=True,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

020: truncation=True,

--> Explication : Paramètres de tokenization pour limiter la longueur des séquences et éviter débordements.

021: max\_length=128,

--> Explication : Paramètres de tokenization pour limiter la longueur des séquences et éviter débordements.

022: padding='max\_length'

--> Explication : Paramètres de tokenization pour limiter la longueur des séquences et éviter débordements.

023: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

024:

--> Explication : Ligne vide (séparateur / lisibilité).

025: offset\_mapping = encoding['offset\_mapping']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

026: labels = ['O'] \* len(offset\_mapping)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

027:

--> Explication : Ligne vide (séparateur / lisibilité).

028: for entity in entities:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

029: start\_char = entity['start']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

030: end\_char = entity['end']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

031: entity\_label = entity['label']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

032:

--> Explication : Ligne vide (séparateur / lisibilité).

033: token\_indices = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

034: for idx, (token\_start, token\_end) in enumerate(offset\_mapping):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

035: if token\_start == token\_end == 0:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

036: continue

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

037: if not (token\_end <= start\_char or token\_start >= end\_char):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

038: token\_indices.append(idx)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

039:

--> Explication : Ligne vide (séparateur / lisibilité).

040: if token\_indices:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

041: labels[token\_indices[0]] = f'B-{entity\_label}'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

042: for idx in token\_indices[1:]:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

043: labels[idx] = f'I-{entity\_label}'

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

044:

--> Explication : Ligne vide (séparateur / lisibilité).

045: label\_ids = [label2id.get(label, 0) for label in labels]

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

046:

--> Explication : Ligne vide (séparateur / lisibilité).

047: return {

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

048: 'input\_ids': encoding['input\_ids'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

049: 'attention\_mask': encoding['attention\_mask'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

050: 'labels': label\_ids

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

051: }

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

052:

--> Explication : Ligne vide (séparateur / lisibilité).

053: # Preparer les datasets NER

--> Explication : Commentaire expliquant l'intention du bloc de code.

054: print('\nConversion en format BIO...')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

055: ner\_train\_data = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

056: for \_, row in train\_df.iterrows():

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

057: entities = row['parsed\_entities'] if row['intent'] == 'TRIP' else []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

058: ner\_example = convert\_to\_bio\_tags(row['text'], entities)

--> Explication : Conversion des annotations en format BIO pour l'entraînement du NER (Token Classification).

059: ner\_train\_data.append(ner\_example)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

060:

--> Explication : Ligne vide (séparateur / lisibilité).

061: ner\_test\_data = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

062: for \_, row in test\_df.iterrows():

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

063: entities = row['parsed\_entities'] if row['intent'] == 'TRIP' else []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

064: ner\_example = convert\_to\_bio\_tags(row['text'], entities)

--> Explication : Conversion des annotations en format BIO pour l'entraînement du NER (Token Classification).

065: ner\_test\_data.append(ner\_example)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

066:

--> Explication : Ligne vide (séparateur / lisibilité).

067: ner\_train\_dataset = Dataset.from\_list(ner\_train\_data)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

068: ner\_test\_dataset = Dataset.from\_list(ner\_test\_data)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

069:

--> Explication : Ligne vide (séparateur / lisibilité).

070: print(f'Datasets NER crees:')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

071: print(f' Train: {len(ner\_train\_dataset)} exemples')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

072: print(f' Test: {len(ner\_test\_dataset)} exemples')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- CamemBERT est choisi pour ses performances sur le français (pré-entraîné sur OSCAR FR).

- La tokenization avec padding/truncation assure des entrées de taille fixe pour le batching.

- Le NER en format BIO permet d'extraire des entités (Departure/Destination) même pour noms composés.

## Cellule 17 - MARKDOWN

Contenu (Markdown) :

## Fine-Tuning NER avec CamemBERT

Entrainement du modele NER pour extraire Departure et Destination de maniere contextuelle.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 18 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from transformers import (

--> Explication : Import ciblé : from transformers import (. Permet d'importer des fonctions/classes spécifiques d'un package.

002: AutoModelForTokenClassification,

--> Explication : Chargement d'un tokenizer / modèle pré-entraîné depuis HuggingFace (CamemBERT), adapté au français.

003: DataCollatorForTokenClassification,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

004: TrainingArguments,

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

005: Trainer

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

006: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

007: import evaluate

--> Explication : Import de module(s) : evaluate. Ces imports fournissent les fonctions/classes nécessaires (p.ex. pandas pour manipulation de données).

008:

--> Explication : Ligne vide (séparateur / lisibilité).

009: # Charger le modele CamemBERT pour Token Classification

--> Explication : Commentaire expliquant l'intention du bloc de code.

010: ner\_model = AutoModelForTokenClassification.from\_pretrained(

--> Explication : Chargement d'un tokenizer / modèle pré-entraîné depuis HuggingFace (CamemBERT), adapté au français.

011: 'camembert-base',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

012: num\_labels=len(ner\_labels),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

013: id2label=id2label,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

014: label2id=label2id

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

015: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

016:

--> Explication : Ligne vide (séparateur / lisibilité).

017: # Data collator

--> Explication : Commentaire expliquant l'intention du bloc de code.

018: data\_collator = DataCollatorForTokenClassification(tokenizer\_fast)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

019:

--> Explication : Ligne vide (séparateur / lisibilité).

020: # Metrique seqeval

--> Explication : Commentaire expliquant l'intention du bloc de code.

021: seqeval\_metric = evaluate.load('seqeval')

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

022:

--> Explication : Ligne vide (séparateur / lisibilité).

023: def compute\_ner\_metrics(eval\_pred):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

024: """Calcule les metriques NER avec seqeval."""

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

025: logits, labels = eval\_pred

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

026: predictions = np.argmax(logits, axis=-1)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

027:

--> Explication : Ligne vide (séparateur / lisibilité).

028: true\_labels = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

029: pred\_labels = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

030:

--> Explication : Ligne vide (séparateur / lisibilité).

031: for pred\_seq, label\_seq in zip(predictions, labels):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

032: true\_tags = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

033: pred\_tags = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

034:

--> Explication : Ligne vide (séparateur / lisibilité).

035: for pred\_id, label\_id in zip(pred\_seq, label\_seq):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

036: if label\_id != -100:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

037: true\_tags.append(id2label[label\_id])

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

038: pred\_tags.append(id2label[pred\_id])

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

039:

--> Explication : Ligne vide (séparateur / lisibilité).

040: true\_labels.append(true\_tags)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

041: pred\_labels.append(pred\_tags)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

042:

--> Explication : Ligne vide (séparateur / lisibilité).

043: results = seqeval\_metric.compute(predictions=pred\_labels, references=true\_labels)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

044:

--> Explication : Ligne vide (séparateur / lisibilité).

045: return {

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

046: 'precision': results['overall\_precision'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

047: 'recall': results['overall\_recall'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

048: 'f1': results['overall\_f1'],

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

049: 'accuracy': results['overall\_accuracy']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

050: }

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

051:

--> Explication : Ligne vide (séparateur / lisibilité).

052: # Arguments d'entrainement

--> Explication : Commentaire expliquant l'intention du bloc de code.

053: ner\_training\_args = TrainingArguments(

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

054: output\_dir=os.path.join(workdir, 'models/ner\_model'),

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

055: num\_train\_epochs=4,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

056: per\_device\_train\_batch\_size=8,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

057: per\_device\_eval\_batch\_size=16,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

058: learning\_rate=3e-5,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

059: weight\_decay=0.01,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

060: eval\_strategy='epoch',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

061: save\_strategy='epoch',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

062: load\_best\_model\_at\_end=True,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

063: metric\_for\_best\_model='f1',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

064: logging\_steps=100,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

065: warmup\_steps=200,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

066: fp16=torch.cuda.is\_available(),

--> Explication : Vérifie la disponibilité d'un GPU CUDA pour accélérer l'entraînement. Si disponible, adapte la configuration (fp16, device).

067: push\_to\_hub=False,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

068: report\_to='none',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

069: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

070:

--> Explication : Ligne vide (séparateur / lisibilité).

071: # Creer le Trainer

--> Explication : Commentaire expliquant l'intention du bloc de code.

072: ner\_trainer = Trainer(

--> Explication : Configuration de l'entraînement : stratégie d'évaluation, taille batch, nombre d'épochs, sauvegarde.

073: model=ner\_model,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

074: args=ner\_training\_args,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

075: train\_dataset=ner\_train\_dataset,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

076: eval\_dataset=ner\_test\_dataset,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

077: tokenizer=tokenizer\_fast,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

078: data\_collator=data\_collator,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

079: compute\_metrics=compute\_ner\_metrics

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

080: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

081:

--> Explication : Ligne vide (séparateur / lisibilité).

082: print('\n' + '='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

083: print('DEBUT DU FINE-TUNING NER')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

084: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

085:

--> Explication : Ligne vide (séparateur / lisibilité).

086: # Entrainer le modele

--> Explication : Commentaire expliquant l'intention du bloc de code.

087: ner\_trainer.train()

--> Explication : Lancement de la boucle d'entraînement du modèle (fine-tuning).

088:

--> Explication : Ligne vide (séparateur / lisibilité).

089: print('\nFine-tuning NER termine!')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- CamemBERT est choisi pour ses performances sur le français (pré-entraîné sur OSCAR FR).

- TrainingArguments centralise la configuration d'entrainement (epoch, batch, lr) pour reproductibilité et réglage fin.

- Le NER en format BIO permet d'extraire des entités (Departure/Destination) même pour noms composés.

- Seqeval est utilisé pour mesurer la qualité du NER (precision/recall/F1 par entité).

- Utilisation de fp16 et GPU si disponible pour accélérer l'entrainement et réduire mémoire.

## Cellule 19 - MARKDOWN

Contenu (Markdown) :

## Evaluation du modele NER

Evalue le modele NER avec metriques detaillees par entite.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 20 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from seqeval.metrics import classification\_report as seqeval\_report

--> Explication : Import ciblé : from seqeval.metrics import classification\_report as seqeval\_report. Permet d'importer des fonctions/classes spécifiques d'un package.

002:

--> Explication : Ligne vide (séparateur / lisibilité).

003: # Evaluer sur le test set

--> Explication : Commentaire expliquant l'intention du bloc de code.

004: ner\_eval\_results = ner\_trainer.evaluate()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

005:

--> Explication : Ligne vide (séparateur / lisibilité).

006: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

007: print('RESULTATS NER SUR LE TEST SET')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

008: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

009: for key, value in ner\_eval\_results.items():

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

010: if isinstance(value, float):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

011: print(f' {key}: {value:.4f}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

012:

--> Explication : Ligne vide (séparateur / lisibilité).

013: # Predictions detaillees

--> Explication : Commentaire expliquant l'intention du bloc de code.

014: ner\_predictions = ner\_trainer.predict(ner\_test\_dataset)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

015: predicted\_logits = ner\_predictions.predictions

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

016: predicted\_labels = np.argmax(predicted\_logits, axis=-1)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

017: true\_labels = ner\_predictions.label\_ids

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

018:

--> Explication : Ligne vide (séparateur / lisibilité).

019: # Convertir en tags pour seqeval

--> Explication : Commentaire expliquant l'intention du bloc de code.

020: true\_tags\_list = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

021: pred\_tags\_list = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

022:

--> Explication : Ligne vide (séparateur / lisibilité).

023: for pred\_seq, label\_seq in zip(predicted\_labels, true\_labels):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

024: true\_tags = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

025: pred\_tags = []

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

026:

--> Explication : Ligne vide (séparateur / lisibilité).

027: for pred\_id, label\_id in zip(pred\_seq, label\_seq):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

028: if label\_id != -100:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

029: true\_tags.append(id2label[label\_id])

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

030: pred\_tags.append(id2label[pred\_id])

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

031:

--> Explication : Ligne vide (séparateur / lisibilité).

032: true\_tags\_list.append(true\_tags)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

033: pred\_tags\_list.append(pred\_tags)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

034:

--> Explication : Ligne vide (séparateur / lisibilité).

035: # Classification report detaille

--> Explication : Commentaire expliquant l'intention du bloc de code.

036: print('\n' + '='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

037: print('CLASSIFICATION REPORT NER (par entite)')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

038: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

039: print(seqeval\_report(true\_tags\_list, pred\_tags\_list, digits=4))

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

040:

--> Explication : Ligne vide (séparateur / lisibilité).

041: # Sauvegarder le modele

--> Explication : Commentaire expliquant l'intention du bloc de code.

042: ner\_model\_path = os.path.join(workdir, 'models/ner\_model\_best')

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

043: ner\_trainer.save\_model(ner\_model\_path)

--> Explication : Sauvegarde du modèle et tokenizer pour réutilisation en production/inference.

044: tokenizer\_fast.save\_pretrained(ner\_model\_path)

--> Explication : Sauvegarde du modèle et tokenizer pour réutilisation en production/inference.

045:

--> Explication : Ligne vide (séparateur / lisibilité).

046: print(f'\nModele NER sauvegarde dans: {ner\_model\_path}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- Le NER en format BIO permet d'extraire des entités (Departure/Destination) même pour noms composés.

- Seqeval est utilisé pour mesurer la qualité du NER (precision/recall/F1 par entité).

## Cellule 21 - MARKDOWN

Contenu (Markdown) :

## Pipeline d'inference complet avec Transformers

Combine Intent Classification + NER pour predictions completes.

Explication :

Ce bloc est une cellule markdown décrivant l'objectif ou le contexte du code suivant. Il aide le lecteur à comprendre le but des cellules de code associées.

## Cellule 22 - CODE

Code (chaque ligne est affichée puis expliquée) :

001: from transformers import pipeline

--> Explication : Import ciblé : from transformers import pipeline. Permet d'importer des fonctions/classes spécifiques d'un package.

002:

--> Explication : Ligne vide (séparateur / lisibilité).

003: # Charger les pipelines

--> Explication : Commentaire expliquant l'intention du bloc de code.

004: print('Chargement des modeles...')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

005:

--> Explication : Ligne vide (séparateur / lisibilité).

006: intent\_model\_path = os.path.join(workdir, 'models/intent\_classifier\_best')

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

007: ner\_model\_path = os.path.join(workdir, 'models/ner\_model\_best')

--> Explication : Construit des chemins de fichiers portables (concaténation safe).

008:

--> Explication : Ligne vide (séparateur / lisibilité).

009: intent\_pipeline = pipeline(

--> Explication : Création d'un pipeline HuggingFace pour faciliter l'inférence (text-classification / token-classification).

010: 'text-classification',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

011: model=intent\_model\_path,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

012: tokenizer=intent\_model\_path,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

013: device=0 if torch.cuda.is\_available() else -1

--> Explication : Vérifie la disponibilité d'un GPU CUDA pour accélérer l'entraînement. Si disponible, adapte la configuration (fp16, device).

014: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

015:

--> Explication : Ligne vide (séparateur / lisibilité).

016: ner\_pipeline = pipeline(

--> Explication : Création d'un pipeline HuggingFace pour faciliter l'inférence (text-classification / token-classification).

017: 'token-classification',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

018: model=ner\_model\_path,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

019: tokenizer=ner\_model\_path,

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

020: aggregation\_strategy='simple',

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

021: device=0 if torch.cuda.is\_available() else -1

--> Explication : Vérifie la disponibilité d'un GPU CUDA pour accélérer l'entraînement. Si disponible, adapte la configuration (fp16, device).

022: )

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

023:

--> Explication : Ligne vide (séparateur / lisibilité).

024: print('Pipelines charges!')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

025:

--> Explication : Ligne vide (séparateur / lisibilité).

026: def predict\_travel\_order(sentence\_id, text):

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

027: """Pipeline complet : Intent + NER"""

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

028: # 1. Predire l'intent

--> Explication : Commentaire expliquant l'intention du bloc de code.

029: intent\_result = intent\_pipeline(text)[0]

--> Explication : Création d'un pipeline HuggingFace pour faciliter l'inférence (text-classification / token-classification).

030: predicted\_intent = intent\_result['label']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

031: confidence = intent\_result['score']

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

032:

--> Explication : Ligne vide (séparateur / lisibilité).

033: # 2. Si pas TRIP, retourner l'intent

--> Explication : Commentaire expliquant l'intention du bloc de code.

034: if predicted\_intent != 'TRIP':

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

035: return f"{sentence\_id},{predicted\_intent}", confidence

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

036:

--> Explication : Ligne vide (séparateur / lisibilité).

037: # 3. Si TRIP, extraire les entites

--> Explication : Commentaire expliquant l'intention du bloc de code.

038: ner\_results = ner\_pipeline(text)

--> Explication : Création d'un pipeline HuggingFace pour faciliter l'inférence (text-classification / token-classification).

039:

--> Explication : Ligne vide (séparateur / lisibilité).

040: departure = None

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

041: destination = None

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

042:

--> Explication : Ligne vide (séparateur / lisibilité).

043: for entity in ner\_results:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

044: entity\_label = entity.get('entity\_group', entity.get('entity'))

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

045: word = entity['word'].replace('▁', ' ').strip()

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

046:

--> Explication : Ligne vide (séparateur / lisibilité).

047: if 'Departure' in entity\_label and not departure:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

048: departure = word

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

049: elif 'Destination' in entity\_label and not destination:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

050: destination = word

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

051:

--> Explication : Ligne vide (séparateur / lisibilité).

052: if departure and destination:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

053: return f"{sentence\_id},{departure},{destination}", confidence

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

054: else:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

055: return f"{sentence\_id},UNKNOWN", confidence

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

056:

--> Explication : Ligne vide (séparateur / lisibilité).

057: # Tester sur les exemples problematiques mentionnes

--> Explication : Commentaire expliquant l'intention du bloc de code.

058: print('\n' + '='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

059: print('TESTS SUR EXEMPLES PROBLEMATIQUES')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

060: print('='\*70)

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

061:

--> Explication : Ligne vide (séparateur / lisibilité).

062: test\_examples = [

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

063: ("4740", "Bonjour, je compte faire le trajet vendredi de Cotonou à Porto-Novo"),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

064: ("5606", "À quelle heure y a-t-il des trains de Martigues à Lyon ?"),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

065: ("4824", "Comment aller de Malanville à Caen ?"),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

066: ("4205", "Is there a train to Manchester?"),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

067: ("3228", "de Semur-en-Auxois à Dreux"),

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

068: ("2745", "Billet Les Sables-d'Olonne à Château-Chinon")

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

069: ]

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

070:

--> Explication : Ligne vide (séparateur / lisibilité).

071: for sid, text in test\_examples:

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

072: result, conf = predict\_travel\_order(sid, text)

--> Explication : Instruction/ligne de code : exécution opérationnelle — voir le code pour le détail exact.

073: print(f'\n{sid}: {text}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

074: print(f' -> Prediction: {result}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

075: print(f' -> Confiance: {conf:.2%}')

--> Explication : Affichage d'information pour suivi/diagnostic lors de l'exécution.

Choix et raisons (pour cette cellule) :

- Le NER en format BIO permet d'extraire des entités (Departure/Destination) même pour noms composés.

- Utilisation de fp16 et GPU si disponible pour accélérer l'entrainement et réduire mémoire.

Document généré automatiquement. Pour questions/demandes d'ajustement, demandez une version plus détaillée pour les cellules spécifiques.