Double-click (or enter) to edit

```
# Importation des librairies
import os
import json
import random
import nltk
import pickle
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset
from tqdm import tqdm
import matplotlib.pyplot as plt
import numpy as np
import joblib
import time

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
#nltk.download('stopwords')
from collections import Counter
from sklearn.metrics import accuracy_score, classification_report
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVC
import threading
from tkinter import *
from tkinter import scrolledtext

print('Toutes les librairies ont été importées avec succès !')
print('='*50)
```

```
Toutes les librairies ont été importées avec succès !
==================================================
```

```
"""# Charger l'ensemble de données
with open('data/healthcare.json','r',encoding='utf-8') as f:
    data = json.load(f)

intents = {'intents':[]}
#
for convo in data:
    tag = convo.get("agent_selected_tool", "general").replace(" ","_").lower()


    pattern = convo.get("user_1", "")
    response = convo.get("agent_initial_response", "")


    if pattern and response:
        intents['intents'].append({
            "tag":tag,
            "patterns":[pattern],
            "responses":[response]
        })

# Save the new dataset
with open("data/healthcare_intents.json", "w", encoding="utf-8") as f:
    json.dump(intents, f, indent=4)
```

```
print("Les intentions de l'ensemble de données ont été créées avec succès !")"""
```

```
Les intentions de l'ensemble de données ont été créées avec succès !
```

```
# Classe principale du Modele
class ChatbotModel(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, 512)
        self.bn1 = nn.BatchNorm1d(512)
        self.fc2 = nn.Linear(512, 256)
        self.bn2 = nn.BatchNorm1d(256)
        self.fc3 = nn.Linear(256, 128)
        self.bn3 = nn.BatchNorm1d(128)
        self.fc4 = nn.Linear(128, 64)
        self.drop = nn.Dropout(0.4)
        self.out = nn.Linear(64, output_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.bn1(self.fc1(x)))
        x = self.drop(x)
        x = self.relu(self.bn2(self.fc2(x)))
        x = self.drop(x)
        x = self.relu(self.bn3(self.fc3(x)))
        x = self.drop(x)
        x = self.relu(self.fc4(x))
        x = self.out(x)
        return x
```

Comparaison entre différents modèles

```
# Assistant chatbot
class ChatbotAssistant:
    def __init__(self, intents_path, function_mappings=None, device=None):
        self.intents_path = intents_path
        self.function_mappings = function_mappings

        self.documents = []
        self.intents = []
        self.intents_responses = {}

        self.vectorizer = None
        self.label_names = None
        self.model = None

        self.X = None
        self.y = None

        self.device = device or ("cuda" if torch.cuda.is_available() else "cpu")

        for r in ["punkt", "wordnet", "omw-1.4", "stopwords"]:
            try:
                nltk.data.find(f"corpora/{r}")
            except LookupError:
                nltk.download(r)


    # Prétraitement du texte
    @staticmethod
```

```
def preprocess_text(text):
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words("english"))
    if not isinstance(text, str):
        return ""
    tokens = nltk.word_tokenize(text.lower())
    tokens = [t for t in tokens if t.isalpha() and t not in stop_words]
    lemmas = [lemmatizer.lemmatize(t) for t in tokens]
    return " ".join(lemmas)



# Analyser les intentions
def parse_intents(self):
    if not os.path.exists(self.intents_path):
        raise FileNotFoundError(f"File not found: {self.intents_path}")

    with open(self.intents_path, "r", encoding="utf-8") as f:
        data = json.load(f)

    self.documents = []
    self.intents = []
    self.intents_responses = {}

    for intent in data.get("intents", []):
        tag = intent.get("tag")
        if tag is None:
            continue
        if tag not in self.intents:
            self.intents.append(tag)
            self.intents_responses[tag] = intent.get("responses", [])

        for pattern in intent.get("patterns", []):
            cleaned = self.preprocess_text(pattern)
            self.documents.append((cleaned, tag))

    print(f"Loaded {len(self.documents)} patterns across {len(self.intents)} intents.")



# Créer des caractéristiques TF-IDF
def build_features(self, max_features=2000):
    texts = [doc[0] for doc in self.documents]
    tags = [doc[1] for doc in self.documents]

    self.label_names = sorted(list(set(self.intents)))
    tag_to_idx = {t: i for i, t in enumerate(self.label_names)}
    self.y = np.array([tag_to_idx[t] for t in tags])

    self.vectorizer = TfidfVectorizer(max_features=max_features)
    self.X = self.vectorizer.fit_transform(texts).toarray()
    print(f"TF-IDF built: X.shape={self.X.shape}, y.shape={self.y.shape}")



# Entraîner et comparer des modèles
def compare_models(self, test_size=0.2, random_state=42, max_features=2000):
    """Compare plusieurs modèles ML sur le même dataset (TF-IDF)."""


    try:
        from xgboost import XGBClassifier
        xgb_available = True
    except ImportError:
        print(" XGBoost not installed, skipping it.")
        xgb_available = False

    # Préparation des données
    if self.X is None or self.y is None:
        self.build_features(max_features=max_features)
    # Filtrer les classes rares (<2 exemples)
    cnt = Counter(self.y)
    valid = [cls for cls, c in cnt.items() if c >= 2]

    if len(valid) < 2:
```

```
    raise ValueError("Il n'y a pas assez de classes valides avec au moins 2 échantillons chacune !")

mask = np.isin(self.y, valid)
self.X = self.X[mask]
self.y = self.y[mask]

print(f"Filtered dataset: {len(self.y)} samples, {len(valid)} valid classes")

X_train, X_test, y_train, y_test = train_test_split(
    self.X, self.y, test_size=test_size, random_state=random_state,
    stratify=self.y if len(set(self.y)) > 1 else None
)

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, n_jobs=-1),
    "SVM": LinearSVC(),
    "Naive Bayes": MultinomialNB(),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "MLP (Sklearn)": MLPClassifier(hidden_layer_sizes=(512, 256, 128),
                                   activation='relu', max_iter=300, random_state=42)
}

if xgb_available:
    models["XGBoost"] = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

results = {}
print("\n Comparaison des modèles...\n")

for name, model in models.items():
    print(f"\n Training {name} ...")
    start = time.time()
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    elapsed = time.time() - start
    results[name] = acc
    print(f" Accuracy: {acc:.4f}  |  Time: {elapsed:.2f}s")
    print(classification_report(y_test, preds, zero_division=0))



# --- Comparaison avec Neural Network (Torch)
print("\n Neural Network (Torch)")
input_size = self.X.shape[1]
output_size = len(self.label_names)
model = ChatbotModel(input_size, output_size)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

X_train_t = torch.tensor(X_train, dtype=torch.float32)
y_train_t = torch.tensor(y_train, dtype=torch.long)
X_test_t = torch.tensor(X_test, dtype=torch.float32)
y_test_t = torch.tensor(y_test, dtype=torch.long)
train_loader = DataLoader(TensorDataset(X_train_t, y_train_t), batch_size=32, shuffle=True)

for epoch in range(50):
    model.train()
    running_loss = 0.0
    for Xb, yb in train_loader:
        optimizer.zero_grad()
        out = model(Xb)
        loss = criterion(out, yb)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}/50 | Loss: {running_loss/len(train_loader):.4f}")

model.eval()
with torch.no_grad():
    preds_nn = torch.argmax(model(X_test_t), dim=1).numpy()

acc_nn = accuracy_score(y_test, preds_nn)
results["Neural Network"] = acc_nn
print("\n Précision des réseaux neuronaux:", acc_nn)
```

```
        unique_labels = np.unique(y_test)
        filtered_names = [self.label_names[i] for i in unique_labels if i < len(self.label_names)]
        print(classification_report(y_test, preds_nn, labels=unique_labels, target_names=filtered_names))

        # Résumé global
        print("\n Résumé de la comparaison des modèles:")
        for k, v in results.items():
            print(f"{k:<25} → {v:.4f}")

        best_model = max(results, key=results.get)
        print(f"\n Best model: {best_model} with accuracy = {results[best_model]:.4f}")

        # Graphique comparatif
        plt.figure(figsize=(10, 6))
        plt.barh(list(results.keys()), list(results.values()), color='skyblue')
        plt.xlabel("Accuracy")
        plt.title("Comparaison des modèles")
        plt.grid(axis="x", linestyle="--", alpha=0.7)
        plt.show()

        return results
```

```
# Exécution
assistant = ChatbotAssistant("data/healthcare_intents.json")
assistant.parse_intents()
assistant.build_features(max_features=2000)

results = assistant.compare_models()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Calixte\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Calixte\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\Calixte\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
Loaded 758 patterns across 137 intents.
TF-IDF built: X.shape=(758, 852), y.shape=(758,)
 XGBoost not installed, skipping it.
Filtered dataset: 690 samples, 69 valid classes

 Comparaison des modèles...


 Training Logistic Regression ...
 Accuracy: 0.6232  |  Time: 47.91s
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           7       0.00      0.00      0.00         1
           9       0.80      1.00      0.89         8
          10       0.00      0.00      0.00         1
          11       0.25      0.40      0.31         5
          12       0.00      0.00      0.00         1
          13       0.67      1.00      0.80         2
          14       0.00      0.00      0.00         1
          16       0.80      1.00      0.89         8
          20       0.00      0.00      0.00         2
          21       1.00      1.00      1.00         1
          26       0.00      0.00      0.00         1
          28       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         1
          31       0.00      0.00      0.00         1
          32       0.50      0.60      0.55         5
          35       0.00      0.00      0.00         2
          36       0.50      0.50      0.50         4
          38       1.00      1.00      1.00         1
          48       0.00      0.00      0.00         1
          49       1.00      1.00      1.00         3
          50       1.00      0.75      0.86         4
          54       0.00      0.00      0.00         1
          55       0.00      0.00      0.00         1
          56       0.75      1.00      0.86         6
          60       0.00      0.00      0.00         1
          63       1.00      0.33      0.50         3
          66       0.00      0.00      0.00         1
          75       0.00      0.00      0.00         1
          76       0.67      1.00      0.80         2
          83       0.00      0.00      0.00         1
          90       0.33      0.50      0.40         2
          95       0.00      0.00      0.00         1
          97       0.80      1.00      0.89         8
          99       0.00      0.00      0.00         1
         101       0.62      1.00      0.77         5
         104       0.00      0.00      0.00         1
         105       0.47      1.00      0.64         9
         108       0.00      0.00      0.00         1
         109       0.00      0.00      0.00         1
         110       0.00      0.00      0.00         1
         111       0.00      0.00      0.00         1
         113       0.00      0.00      0.00         2
         114       0.00      0.00      0.00         1
         115       0.00      0.00      0.00         1
         116       1.00      1.00      1.00         3
         118       1.00      0.25      0.40         4
         122       0.00      0.00      0.00         1
         127       0.46      1.00      0.63        13
         128       0.00      0.00      0.00         1
         130       1.00      0.67      0.80         3
         132       0.00      0.00      0.00         2
         135       1.00      1.00      1.00         2

    accuracy                           0.62       138
   macro avg       0.30      0.33      0.30       138
weighted avg       0.51      0.62      0.53       138
```

```
Training SVM ...
Accuracy: 0.6812  |  Time: 0.06s
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           7       1.00      1.00      1.00         1
           9       0.89      1.00      0.94         8
          10       0.00      0.00      0.00         1
          11       0.29      0.40      0.33         5
          12       0.00      0.00      0.00         1
          13       0.67      1.00      0.80         2
          14       0.00      0.00      0.00         1
          16       0.89      1.00      0.94         8
          20       1.00      1.00      1.00         2
          21       1.00      1.00      1.00         1
          26       0.50      1.00      0.67         1
          28       0.00      0.00      0.00         1
          30       0.50      1.00      0.67         1
          31       0.00      0.00      0.00         1
          32       0.60      0.60      0.60         5
          33       0.00      0.00      0.00         0
          35       0.00      0.00      0.00         2
          36       0.40      0.50      0.44         4
          38       1.00      1.00      1.00         1
          48       0.00      0.00      0.00         1
          49       1.00      1.00      1.00         3
          50       0.67      1.00      0.80         4
          54       0.00      0.00      0.00         1
          55       0.00      0.00      0.00         1
          56       1.00      1.00      1.00         6
          60       0.00      0.00      0.00         1
          63       1.00      0.67      0.80         3
          66       0.00      0.00      0.00         1
          75       0.00      0.00      0.00         1
          76       0.67      1.00      0.80         2
          83       0.00      0.00      0.00         1
          90       0.50      0.50      0.50         2
          95       0.00      0.00      0.00         1
          97       0.80      1.00      0.89         8
          99       0.00      0.00      0.00         1
         101       0.62      1.00      0.77         5
         104       0.00      0.00      0.00         1
         105       0.69      1.00      0.82         9
         108       0.00      0.00      0.00         1
         109       0.00      0.00      0.00         1
         110       0.00      0.00      0.00         1
         111       0.00      0.00      0.00         1
         112       0.00      0.00      0.00         0
         113       0.50      0.50      0.50         2
         114       0.00      0.00      0.00         1
         115       0.00      0.00      0.00         1
         116       1.00      1.00      1.00         3
         118       0.50      0.25      0.33         4
         122       1.00      1.00      1.00         1
         127       1.00      0.85      0.92        13
         128       0.00      0.00      0.00         1
         130       1.00      0.67      0.80         3
         132       0.50      0.50      0.50         2
         135       0.50      1.00      0.67         2

    accuracy                           0.68       138
   macro avg       0.38      0.43      0.39       138
weighted avg       0.62      0.68      0.64       138


Training Naive Bayes ...
Accuracy: 0.4710  |  Time: 0.06s
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           7       0.00      0.00      0.00         1
           9       0.73      1.00      0.84         8
          10       0.00      0.00      0.00         1
          11       0.50      0.40      0.44         5
          12       0.00      0.00      0.00         1
          13       0.00      0.00      0.00         2
          14       0.00      0.00      0.00         1
```

```
16       0.67      1.00      0.80         8
20       0.00      0.00      0.00         2
21       0.00      0.00      0.00         1
26       0.00      0.00      0.00         1
28       0.00      0.00      0.00         1
30       0.00      0.00      0.00         1
31       0.00      0.00      0.00         1
32       1.00      0.20      0.33         5
35       0.00      0.00      0.00         2
36       0.00      0.00      0.00         4
38       0.00      0.00      0.00         1
48       0.00      0.00      0.00         1
49       1.00      0.33      0.50         3
50       1.00      0.25      0.40         4
54       0.00      0.00      0.00         1
55       0.00      0.00      0.00         1
56       1.00      0.83      0.91         6
60       0.00      0.00      0.00         1
63       0.00      0.00      0.00         3
66       0.00      0.00      0.00         1
75       0.00      0.00      0.00         1
76       0.67      1.00      0.80         2
83       0.00      0.00      0.00         1
90       0.00      0.00      0.00         2
95       0.00      0.00      0.00         1
97       0.73      1.00      0.84         8
99       0.00      0.00      0.00         1
101       0.62      1.00      0.77         5
104       0.00      0.00      0.00         1
105       0.36      1.00      0.53         9
108       0.00      0.00      0.00         1
109       0.00      0.00      0.00         1
110       0.00      0.00      0.00         1
111       0.00      0.00      0.00         1
113       0.00      0.00      0.00         2
114       0.00      0.00      0.00         1
115       0.00      0.00      0.00         1
116       1.00      0.67      0.80         3
118       0.00      0.00      0.00         4
122       0.00      0.00      0.00         1
127       0.24      1.00      0.39        13
128       0.00      0.00      0.00         1
130       0.00      0.00      0.00         3
132       0.00      0.00      0.00         2
135       0.00      0.00      0.00         2

   accuracy                           0.47       138
  macro avg       0.17      0.18      0.15       138
weighted avg       0.37      0.47      0.36       138


Training Random Forest ...
Accuracy: 0.6594  |  Time: 2.70s
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.50      1.00      0.67         1
           7       0.00      0.00      0.00         1
           9       1.00      1.00      1.00         8
          10       0.00      0.00      0.00         1
          11       0.50      0.60      0.55         5
          12       0.00      0.00      0.00         1
          13       0.67      1.00      0.80         2
          14       0.00      0.00      0.00         1
          16       1.00      1.00      1.00         8
          20       0.00      0.00      0.00         2
          21       1.00      1.00      1.00         1
          26       0.00      0.00      0.00         1
          28       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         1
          31       0.00      0.00      0.00         1
          32       0.50      0.60      0.55         5
          35       0.67      1.00      0.80         2
          36       0.67      0.50      0.57         4
          38       1.00      1.00      1.00         1
          48       0.00      0.00      0.00         1
          49       1.00      1.00      1.00         3
          50       0.16      1.00      0.28         4
          54       0.00      0.00      0.00         1
          55       0.00      0.00      0.00         1
          56       1.00      0.83      0.91         6
```

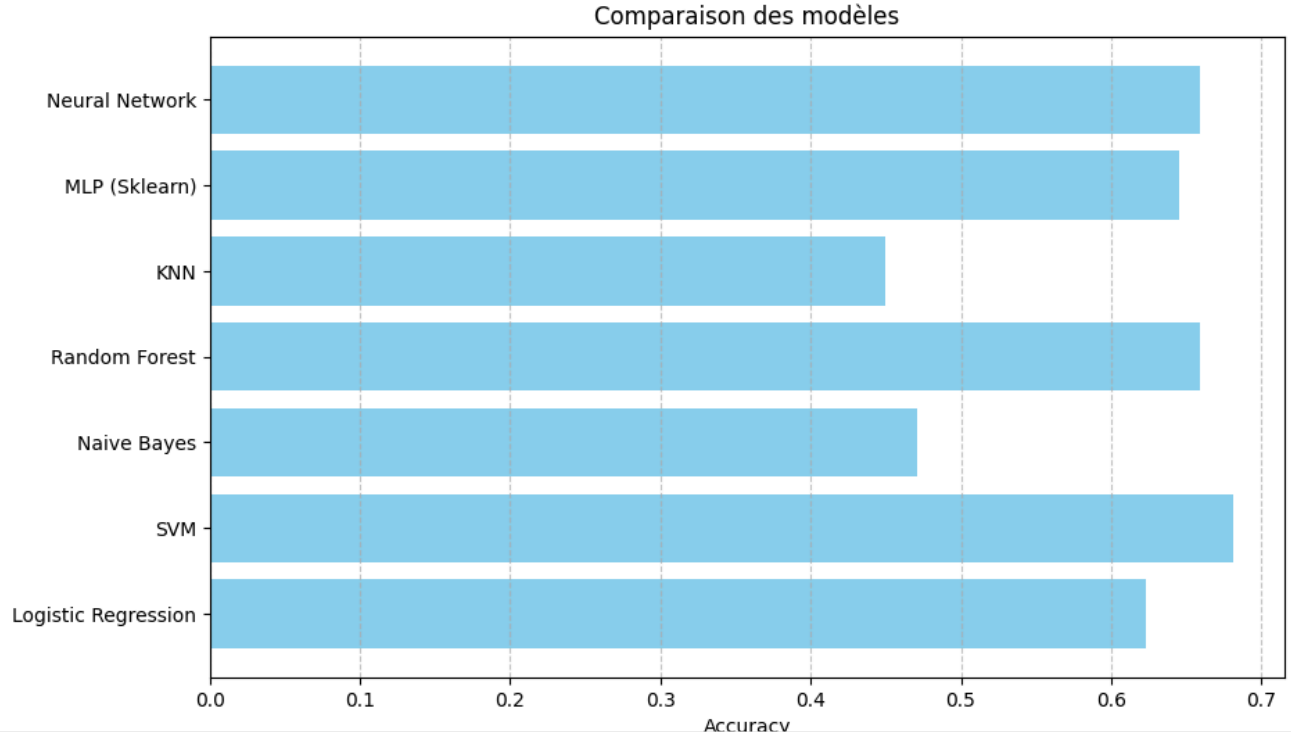
```
              60       0.00      0.00      0.00         1
              63       1.00      0.67      0.80         3
              66       0.00      0.00      0.00         1
              75       0.00      0.00      0.00         1
              76       0.67      1.00      0.80         2
              83       0.00      0.00      0.00         1
              90       0.00      0.00      0.00         2
              95       0.00      0.00      0.00         1
              97       0.80      1.00      0.89         8
              99       0.00      0.00      0.00         1
             101       0.83      1.00      0.91         5
             104       0.00      0.00      0.00         1
             105       0.82      1.00      0.90         9
             108       0.00      0.00      0.00         1
             109       0.00      0.00      0.00         1
             110       0.00      0.00      0.00         1
             111       0.00      0.00      0.00         1
             113       1.00      0.50      0.67         2
             114       0.00      0.00      0.00         1
             115       1.00      1.00      1.00         1
             116       1.00      1.00      1.00         3
             118       0.00      0.00      0.00         4
             122       1.00      1.00      1.00         1
             127       1.00      0.77      0.87        13
             128       1.00      1.00      1.00         1
             130       0.67      0.67      0.67         3
             132       1.00      0.50      0.67         2
             135       0.67      1.00      0.80         2

        accuracy                           0.66       138
       macro avg       0.40      0.43      0.40       138
    weighted avg       0.63      0.66      0.63       138


    Training KNN ...
    Accuracy: 0.4493  |  Time: 0.02s
                  precision    recall  f1-score   support

               1       0.00      0.00      0.00         1
               2       0.50      1.00      0.67         1
               3       0.00      0.00      0.00         1
               7       0.00      0.00      0.00         1
               9       0.83      0.62      0.71         8
              10       0.00      0.00      0.00         1
              11       0.75      0.60      0.67         5
              12       0.00      0.00      0.00         1
              13       1.00      0.50      0.67         2
              14       0.00      0.00      0.00         1
              16       0.86      0.75      0.80         8
              20       0.00      0.00      0.00         2
              21       1.00      1.00      1.00         1
              26       1.00      1.00      1.00         1
              28       1.00      1.00      1.00         1
              30       0.00      0.00      0.00         1
              31       0.00      0.00      0.00         1
              32       0.67      0.40      0.50         5
              33       0.00      0.00      0.00         0
              35       0.00      0.00      0.00         2
              36       0.40      0.50      0.44         4
              38       1.00      1.00      1.00         1
              48       0.00      0.00      0.00         1
              49       1.00      0.67      0.80         3
              50       0.06      1.00      0.11         4
              54       0.00      0.00      0.00         1
              55       0.00      0.00      0.00         1
              56       1.00      0.67      0.80         6
              60       0.00      0.00      0.00         1
              63       1.00      0.33      0.50         3
              66       0.00      0.00      0.00         1
              75       0.00      0.00      0.00         1
              76       0.67      1.00      0.80         2
              83       0.00      0.00      0.00         1
              90       0.00      0.00      0.00         2
              95       0.00      0.00      0.00         1
              97       0.88      0.88      0.88         8
              99       0.00      0.00      0.00         1
             101       0.67      0.40      0.50         5
             104       0.00      0.00      0.00         1
             105       1.00      0.44      0.62         9
             108       0.00      0.00      0.00         1
             109       0.00      0.00      0.00         1
             110       0.00      0.00      0.00         1
```

```
           111       0.00      0.00      0.00         1
           113       0.00      0.00      0.00         2
           114       0.00      0.00      0.00         1
           115       0.00      0.00      0.00         1
           116       1.00      1.00      1.00         3
           118       0.00      0.00      0.00         4
           122       1.00      1.00      1.00         1
           127       1.00      0.62      0.76        13
           128       0.00      0.00      0.00         1
           130       0.00      0.00      0.00         3
           132       0.00      0.00      0.00         2
           135       0.00      0.00      0.00         2

      accuracy                           0.45       138
     macro avg       0.33      0.29      0.29       138
  weighted avg       0.57      0.45      0.47       138


 Training MLP (Sklearn) ...
 Accuracy: 0.6449  |  Time: 33.46s
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.33      1.00      0.50         1
           7       1.00      1.00      1.00         1
           9       0.89      1.00      0.94         8
          10       0.00      0.00      0.00         1
          11       0.40      0.40      0.40         5
          12       0.00      0.00      0.00         1
          13       0.67      1.00      0.80         2
          14       0.00      0.00      0.00         1
          16       1.00      0.88      0.93         8
          20       0.00      0.00      0.00         2
          21       1.00      1.00      1.00         1
          26       1.00      1.00      1.00         1
          28       0.00      0.00      0.00         1
          30       0.00      0.00      0.00         1
          31       0.00      0.00      0.00         1
          32       0.67      0.40      0.50         5
          33       0.00      0.00      0.00         0
          35       0.50      1.00      0.67         2
          36       1.00      0.50      0.67         4
          38       1.00      1.00      1.00         1
          48       0.00      0.00      0.00         1
          49       1.00      1.00      1.00         3
          50       0.31      1.00      0.47         4
          54       0.00      0.00      0.00         1
          55       0.00      0.00      0.00         1
          56       1.00      0.83      0.91         6
          60       0.00      0.00      0.00         1
          63       1.00      0.67      0.80         3
          66       0.00      0.00      0.00         1
          75       0.00      0.00      0.00         1
          76       0.67      1.00      0.80         2
          83       0.00      0.00      0.00         1
          90       1.00      0.50      0.67         2
          95       0.00      0.00      0.00         1
          97       0.78      0.88      0.82         8
          99       0.00      0.00      0.00         1
         101       0.56      1.00      0.71         5
         104       0.00      0.00      0.00         1
         105       0.78      0.78      0.78         9
         108       0.00      0.00      0.00         1
         109       0.00      0.00      0.00         1
         110       0.00      0.00      0.00         1
         111       0.00      0.00      0.00         1
         112       0.00      0.00      0.00         0
         113       1.00      1.00      1.00         2
         114       0.00      0.00      0.00         1
         115       1.00      1.00      1.00         1
         116       1.00      1.00      1.00         3
         118       1.00      0.25      0.40         4
         122       0.00      0.00      0.00         1
         127       1.00      0.77      0.87        13
         128       0.50      1.00      0.67         1
         130       1.00      0.67      0.80         3
         132       0.50      0.50      0.50         2
         135       0.67      1.00      0.80         2

      accuracy                           0.64       138
     macro avg       0.42      0.44      0.41       138
```

```
weighted avg       0.67      0.64      0.63       138


 Neural Network (Torch)
Epoch 1/50 | Loss: 4.8138
Epoch 2/50 | Loss: 4.1486
Epoch 3/50 | Loss: 3.3872
Epoch 4/50 | Loss: 2.8993
Epoch 5/50 | Loss: 2.4934
Epoch 6/50 | Loss: 2.2443
Epoch 7/50 | Loss: 1.9888
Epoch 8/50 | Loss: 1.7744
Epoch 9/50 | Loss: 1.5907
Epoch 10/50 | Loss: 1.4370
Epoch 11/50 | Loss: 1.3119
Epoch 12/50 | Loss: 1.1457
Epoch 13/50 | Loss: 1.0349
Epoch 14/50 | Loss: 0.9988
Epoch 15/50 | Loss: 0.8673
Epoch 16/50 | Loss: 0.8320
Epoch 17/50 | Loss: 0.7864
Epoch 18/50 | Loss: 0.6770
Epoch 19/50 | Loss: 0.6262
Epoch 20/50 | Loss: 0.5919
Epoch 21/50 | Loss: 0.5391
Epoch 22/50 | Loss: 0.4847
Epoch 23/50 | Loss: 0.4099
Epoch 24/50 | Loss: 0.4347
Epoch 25/50 | Loss: 0.4240
Epoch 26/50 | Loss: 0.3890
Epoch 27/50 | Loss: 0.3846
Epoch 28/50 | Loss: 0.3319
Epoch 29/50 | Loss: 0.3511
Epoch 30/50 | Loss: 0.3172
Epoch 31/50 | Loss: 0.3422
Epoch 32/50 | Loss: 0.3077
Epoch 33/50 | Loss: 0.2834
Epoch 34/50 | Loss: 0.2593
Epoch 35/50 | Loss: 0.2795
Epoch 36/50 | Loss: 0.2693
Epoch 37/50 | Loss: 0.1993
Epoch 38/50 | Loss: 0.2435
Epoch 39/50 | Loss: 0.2598
Epoch 40/50 | Loss: 0.2200
Epoch 41/50 | Loss: 0.1952
Epoch 42/50 | Loss: 0.1741
Epoch 43/50 | Loss: 0.1915
Epoch 44/50 | Loss: 0.1683
Epoch 45/50 | Loss: 0.1979
Epoch 46/50 | Loss: 0.2558
Epoch 47/50 | Loss: 0.2013
Epoch 48/50 | Loss: 0.1933
Epoch 49/50 | Loss: 0.2082
Epoch 50/50 | Loss: 0.1703

 Précision des réseaux neuronaux: 0.6594202898550725
                                precision    recall  f1-score   support

            appointment_booking       0.00      0.00      0.00         1
       appointment_cancellation       0.00      0.00      0.00         1
         appointment_reschedule       0.50      1.00      0.67         1
                        billing       1.00      1.00      1.00         1
                billingsupport       1.00      1.00      1.00         8
              book_appointment       0.00      0.00      0.00         1
               bookappointment       0.67      0.40      0.50         5
            cancel_appointment       0.00      0.00      0.00         1
             cancelappointment       0.67      1.00      0.80         2
                check_symptoms       0.00      0.00      0.00         1
            checkreportstatus       1.00      0.75      0.86         8
    confidentialitypolicylookup       1.00      1.00      1.00         2
                confirmupdate       1.00      1.00      1.00         1
              diet_nutrition       0.50      1.00      0.67         1
    dischargeinstructionslookup       0.00      0.00      0.00         1
        doctor_availability       0.00      0.00      0.00         1
      doctor_specialization       0.00      0.00      0.00         1
           doctorinfolookup       0.60      0.60      0.60         5
                  emergency       0.50      1.00      0.67         2
           emergencycontact       0.50      0.25      0.33         4
                   feedback       1.00      1.00      1.00         1
         general_health_tips       0.00      0.00      0.00         1
                     goodbye       0.75      1.00      0.86         3
                    greeting       0.40      1.00      0.57         4
```

insurance	0.00	0.00	0.00	1
insurance_info	0.00	0.00	0.00	1
insuranceclaimsupport	0.75	1.00	0.86	6
insuranceverification	0.00	0.00	0.00	1
labtestbooking	1.00	0.67	0.80	3
location	0.00	0.00	0.00	1
mental_health	0.00	0.00	0.00	1
mentalhealthresourcelookup	0.33	0.50	0.40	2
opening_days	0.00	0.00	0.00	1
pediatricvaccinationschedulelookup	0.50	0.50	0.50	2
prescription_refill	0.00	0.00	0.00	1
prescriptionrefill	0.80	1.00	0.89	8
recordstransferrequest	0.00	0.00	0.00	1
recordtransferrequest	0.83	1.00	0.91	5
reschedule_appointment	0.00	0.00	0.00	1
rescheduleappointment	0.88	0.78	0.82	9
symptom_check	0.00	0.00	0.00	1
symptom_cold	0.00	0.00	0.00	1
symptom_cough	0.00	0.00	0.00	1
symptom_fever	0.00	0.00	0.00	1
telehealthavailability	0.50	0.50	0.50	2
telehealthplatform	0.00	0.00	0.00	1
test_results	1.00	1.00	1.00	1
thanks	1.00	1.00	1.00	3
transfercall	1.00	0.25	0.40	4
travelclearanceformlookup	1.00	1.00	1.00	1
updatepatientinfo	1.00	0.77	0.87	13
vaccination	0.00	0.00	0.00	1
vaccinationschedulelookup	0.50	1.00	0.67	3
vaccinesideeffectsinfo	0.50	0.50	0.50	2
working_hours	0.67	1.00	0.80	2
micro avg	0.67	0.66	0.66	138
macro avg	0.42	0.46	0.43	138
weighted avg	0.66	0.66	0.64	138

```
 Résumé de la comparaison des modèles:
Logistic Regression       → 0.6232
SVM                       → 0.6812
Naive Bayes               → 0.4710
Random Forest             → 0.6594
KNN                       → 0.4493
MLP (Sklearn)             → 0.6449
Neural Network            → 0.6594

 Best model: SVM with accuracy = 0.6812
c:\Users\Calixte\Documents\Capstone-Chatbot-Intelligent\.venv\Lib\site-packages\sklearn\metrics\_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
c:\Users\Calixte\Documents\Capstone-Chatbot-Intelligent\.venv\Lib\site-packages\sklearn\metrics\_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
c:\Users\Calixte\Documents\Capstone-Chatbot-Intelligent\.venv\Lib\site-packages\sklearn\metrics\_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
```



Choix du meilleur modèle

Après avoir comparé plusieurs modèles sur la base de leur taux de précision (Accuracy), nous avons décidé d'entraîner
Le SVM a été retenu car il a obtenu la meilleure précision globale parmi les modèles testés.
Le Neural Network, quant à lui, a été conservé puisque ses performances étaient proches de celles du SVM, et qu'il re

```
# Classe principale du Chatbot

class ChatbotAssistant:
    def __init__(self, intents_path, function_mappings=None, device=None):
        self.intents_path = intents_path
        self.function_mappings = function_mappings

        self.documents = []
        self.intents = []
        self.intents_responses = {}


        self.vectorizer = None
        self.label_names = None
        self.model = None


        self.X = None
        self.y = None


        self.device = device or ("cuda" if torch.cuda.is_available() else "cpu")


    # Prétraitement du texte
    @staticmethod
    def preprocess_text(text):
        lemmatizer = WordNetLemmatizer()
        stop_words = set(stopwords.words('english'))
        if not isinstance(text, str):
            return ""
        tokens = nltk.word_tokenize(text.lower())
        tokens = [t for t in tokens if t.isalpha() and t not in stop_words]
        lemmas = [lemmatizer.lemmatize(t) for t in tokens]
        return " ".join(lemmas)



    # Chargement du fichier intents.json
    def parse_intents(self):
        if not os.path.exists(self.intents_path):
            raise FileNotFoundError(f"File not found: {self.intents_path}")

        with open(self.intents_path, "r", encoding="utf-8") as f:
            data = json.load(f)

        self.documents = []
        self.intents = []
        self.intents_responses = {}

        for intent in data.get("intents", []):
            tag = intent.get("tag")
            if tag is None:
                continue
            if tag not in self.intents:
                self.intents.append(tag)
                self.intents_responses[tag] = intent.get("responses", [])

            for pattern in intent.get("patterns", []):
                cleaned = self.preprocess_text(pattern)

                self.documents.append((cleaned, tag))
```

```
        print(f"Loaded {len(self.documents)} patterns across {len(self.intents)} intents.")



    #  Vectorisation TF-IDF
    def build_features(self, max_features=2000):
        texts = [doc[0] for doc in self.documents]
        tags = [doc[1] for doc in self.documents]


        self.label_names = sorted(list(set(self.intents)))
        tag_to_idx = {t: i for i, t in enumerate(self.label_names)}
        self.y = np.array([tag_to_idx[t] for t in tags])


        self.vectorizer = TfidfVectorizer(max_features=max_features)
        X = self.vectorizer.fit_transform(texts).toarray()
        self.X = X
        print(f"TF-IDF built: X.shape = {self.X.shape}, y.shape = {self.y.shape}")

        cnt = Counter(self.y)
        valid = [cls for cls, c in cnt.items() if c >= 2]
        mask = np.isin(self.y, valid)
        self.X = self.X[mask]
        self.y = self.y[mask]
        print("Filtered shapes:", self.X.shape, self.y.shape)

        old_label_names = self.label_names
        new_label_names = [old_label_names[i] for i in valid]

        old_to_new = {old_idx: new_idx for new_idx, old_idx in enumerate(valid)}

        self.y = np.array([old_to_new[int(old)] for old in self.y if int(old) in old_to_new])

        self.label_names = new_label_names

        print(f"Nouvelle taille de label_names : {len(self.label_names)}")


    #  Entraînement du modèle
    def train_model(self,
                    batch_size=32,
                    lr=1e-3,
                    epochs=50,
                    max_features=2000,
                    test_size=0.2,
                    scheduler_step=None,
                    scheduler_gamma=0.8,
                    random_state=42):

        if self.X is None or self.y is None:
            print("Building features (TF-IDF)...")
            self.build_features(max_features=max_features)

    # Filtrer les classes rares (<2 exemples)
        cnt = Counter(self.y)
        valid = [cls for cls, c in cnt.items() if c >= 2]

        if len(valid) < 2:
            raise ValueError("Il n'y a pas assez de classes valides avec au moins 2 échantillons chacune !")

        mask = np.isin(self.y, valid)
        self.X = self.X[mask]
        self.y = self.y[mask]

        print(f"Filtered dataset: {len(self.y)} samples, {len(valid)} valid classes")

        #   Split train/test
        X_train, X_test, y_train, y_test = train_test_split(
            self.X, self.y, test_size=test_size, random_state=random_state,
            stratify=self.y if len(set(self.y)) > 1 else None
    )
        print(f"Train samples: {len(X_train)}, Test samples: {len(X_test)}")
```

```
        # Données Tensor
        X_train_t = torch.tensor(X_train, dtype=torch.float32)
        y_train_t = torch.tensor(y_train, dtype=torch.long)
        X_test_t = torch.tensor(X_test, dtype=torch.float32)
        y_test_t = torch.tensor(y_test, dtype=torch.long)

        train_loader = DataLoader(TensorDataset(X_train_t, y_train_t),
                                  batch_size=batch_size, shuffle=True)

        # Création du modèle
        input_size = self.X.shape[1]
        output_size = len(valid)

        self.model = ChatbotModel(input_size, output_size)

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(self.model.parameters(), lr=lr)
        scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=scheduler_step, gamma=scheduler_gamma)

        # Entraînement
        print("\n Starting training...\n")
        loss_values = []

        for epoch in range(epochs):
            self.model.train()
            running_loss = 0.0

            for batch_X, batch_y in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs}", leave=False):
                optimizer.zero_grad()
                outputs = self.model(batch_X)
                loss = criterion(outputs, batch_y)
                loss.backward()
                optimizer.step()
                running_loss += loss.item()

            scheduler.step()
            epoch_loss = running_loss / len(train_loader)
            loss_values.append(epoch_loss)
            print(f"Epoch {epoch+1}/{epochs} | Loss: {epoch_loss:.4f} | LR: {scheduler.get_last_lr()[0]:.6f}")

        # Loss Graphe
        plt.figure(figsize=(8, 5))
        plt.plot(loss_values, label="Training Loss", color="blue")
        plt.title("Training Loss Curve")
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.legend()
        plt.grid(True)
        plt.show()

        # Evaluation
        self.model.eval()
        with torch.no_grad():
            preds = torch.argmax(self.model(X_test_t), dim=1).numpy()

        acc = accuracy_score(y_test, preds)
        print("\n=== Test set evaluation ===")
        print(f"Accuracy: {acc:.4f}")

    #   Mettre à jour les noms d'intents valides
        unique_labels = np.unique(y_test)
        filtered_names = [self.label_names[i] for i in unique_labels if i < len(self.label_names)]

        print(classification_report(
        y_test, preds,
        labels=unique_labels,
        target_names=filtered_names,
        zero_division=0
    ))

        print("\n Model training + evaluation complete.")


        print("Model training + evaluation complete.")
```

```
        unique_labels = np.unique(y_test)
        filtered_names = [self.label_names[i] for i in unique_labels]


    # Sauvegarde / Chargement
    def save_all(self, model_path="chat_model.pth", meta_path="meta.pkl"):
        torch.save(self.model.state_dict(), model_path)
        meta = {
            "label_names": self.label_names,
            "vectorizer": self.vectorizer
        }

        with open(meta_path, "wb") as f:
            pickle.dump(meta, f)
        print(f"Saved model to {model_path} and meta to {meta_path}.")

    def load_all(self, model_path="chat_model_tfidf.pth", meta_path="meta_tfidf.pkl"):
        with open(meta_path, "rb") as f:
            meta = pickle.load(f)
        self.label_names = meta["label_names"]
        self.vectorizer = meta["vectorizer"]
        input_size = self.vectorizer.max_features if hasattr(self.vectorizer, "max_features") and self.vectorizer.

        input_size = len(self.vectorizer.get_feature_names_out())
        output_size = len(self.label_names)
        self.model = ChatbotModel(input_size, output_size).to(self.device)
        self.model.load_state_dict(torch.load(model_path, map_location=self.device))
        self.model.eval()
        print(f"Loaded model and meta from {model_path}, {meta_path}.")



    # Interaction avec l'utilisateur
    def process_message(self, input_message, threshold=0.65, log_uncertain=True):
        if self.model is None or self.vectorizer is None:
            raise RuntimeError("Modèle ou vectoriseur non chargé. Appelez d'abord train_model() ou load_all().")

        cleaned = self.preprocess_text(input_message)
        vec = self.vectorizer.transform([cleaned]).toarray()
        input_t = torch.tensor(vec, dtype=torch.float32).to(self.device)

        with torch.no_grad():
            logits = self.model(input_t)
            probs = F.softmax(logits, dim=1)
            confidence, idx = torch.max(probs, dim=1)
            confidence = confidence.item()
            pred_idx = idx.item()
            predicted_intent = self.label_names[pred_idx]

        if confidence >= threshold:

            resp = random.choice(self.intents_responses.get(predicted_intent, ["Sorry, I don't have a response."])
            return f"({confidence*100:.1f}% confident) {resp}"
        else:
            if log_uncertain:
                with open("uncertain_inputs.log", "a", encoding="utf-8") as f:
                    f.write(input_message.strip() + "\n")
            return "I'm not sure I understood that. Could you rephrase?"
```

```
# Exécution

if __name__ == "__main__":
    assistant = ChatbotAssistant("data/healthcare_intents.json")
    assistant.parse_intents()

    assistant.train_model(batch_size=32, lr=1e-3, epochs=50, max_features=2000, test_size=0.2, scheduler_step=15,
    #assistant.save_all("chat_model_tfidf.pth", "meta_tfidf.pkl")

    assistant.load_all('chat_model_tfidf.pth','meta_tfidf.pkl')
```