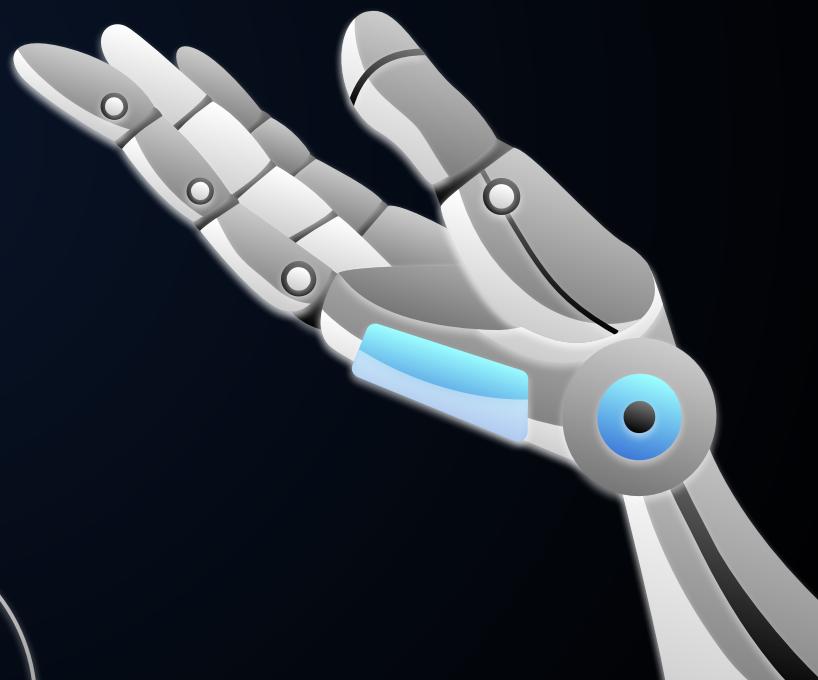
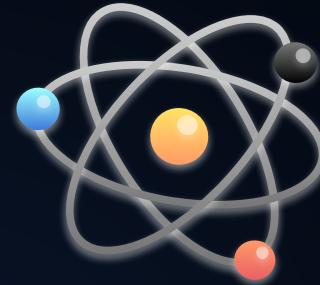
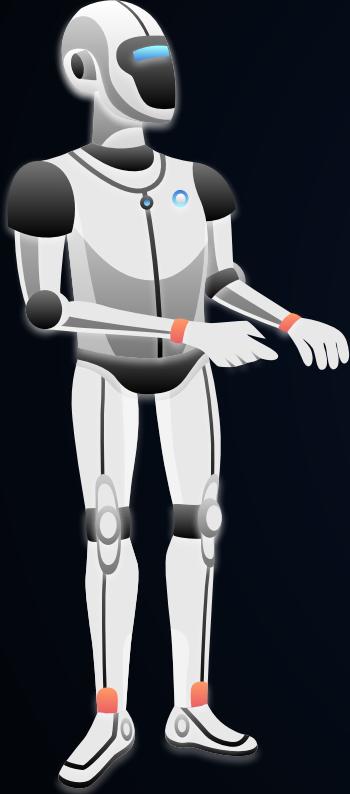


Création d'un Chatbot Multilingue avec NLP et Réseaux Neuronaux



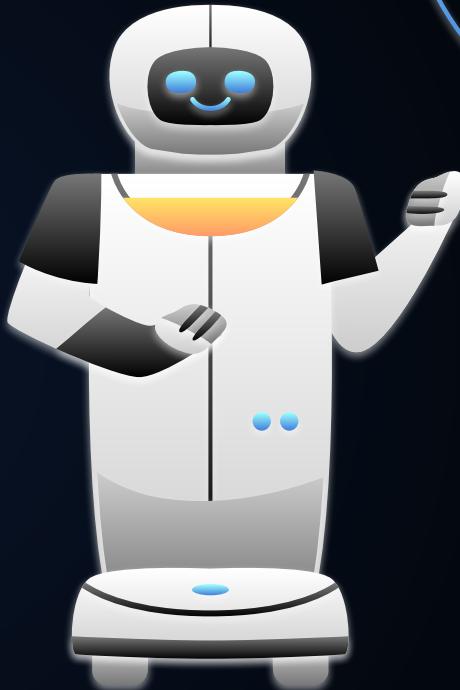


INTRODUCTION

Le chatbot multilingue utilise le traitement du langage naturel (NLP) et les réseaux neuronaux pour interagir avec les utilisateurs dans différentes langues. Il offre une expérience utilisateur améliorée et une communication efficace.

Streamlit

Streamlit est une bibliothèque open-source en Python qui permet de créer facilement des applications web interactives pour le Machine Learning et la Data Science. Elle simplifie le processus de développement en permettant aux développeurs de créer des interfaces utilisateurs élégantes et intuitives avec un minimum de code



```
# Streamlit interface  
st.title('Chatbot Interface with Translation')
```

La création de l'interface avec un titre

```
# Text input pour l'utilisateur  
input_text = st.text_input("Vous:")
```

Création d'un espace texte

```
# Bouton pour envoyer le message  
if st.button("Envoyer"):  
    # Ajout de la question de l'utilisateur à l'historique  
    st.session_state.chat_log.append(("Vous", input_text))
```

Creation de bouton

Chatbot Interface with Translation

Vous:

Envoyer

Traitement du Langage Naturel (NLP)

Le NLP permet au chatbot de comprendre et d'interpréter le langage humain de manière sémantique. Il utilise des algorithmes avancés pour analyser et répondre aux requêtes des utilisateurs de manière contextuelle.



un fichier JSON est utilisé dans le processus d'entraînement du modèle, qui a un format spécifique adapté à la tâche de Natural Language Processing (NLP)

La structure de fichier Json :

- La clé "tag" identifie la catégorie ou l'intention associée à ces motifs de texte.
- La clé "patterns" contient une liste de motifs de texte qui sont associés à la catégorie spécifiée. Ce sont les entrées que le modèle va utiliser pour identifier la catégorie
- La clé "responses" contient une liste de réponses associées à la catégorie. Lorsque le modèle identifie la catégorie, il sélectionne une réponse dans cette liste pour retourner à l'utilisateur.

```
# Tokenisation et suppression de la ponctuation
tokens = nltk.word_tokenize(pattern.translate(punctuation_table))
# Filtrage des mots vides et tokenisation
filtered_words = [word for word in tokens if word.lower() not in stop_words and word not in string.punctuation]
# Application du stemming sur les mots filtrés
stemmed_words = [stemmer.stem(word.lower()) for word in filtered_words]
```

1. Tokenization : La tokenisation est le processus de division du texte en mots ou tokens. Dans notre cas, nous utilisons la fonction `word_tokenize` de la bibliothèque NLTK pour diviser chaque motif de phrase en mots individuels.
2. Suppression de la ponctuation : Après la tokenization, nous utilisons la méthode `translate` pour supprimer tous les signes de ponctuation du texte.
3. Filtrage des mots vides : Les mots vides, ou stop words, sont des mots très courants qui n'apportent généralement pas beaucoup de sens au texte. Nous utilisons une liste prédéfinie de mots vides fournie par NLTK pour filtrer ces mots du texte.
4. Stemming : Le stemming est le processus de réduction des mots à leur forme racine ou base, généralement en supprimant les suffixes. Dans notre cas, nous utilisons l'algorithme de stemming de Porter inclus dans NLTK pour ramener chaque mot à sa forme de base.

```
for x, doc in enumerate(docs_x):
    # Création du sac de mots pour chaque document
    bag = [1 if w in doc else 0 for w in words]
    # Création du vecteur de sortie correspondant
    output_row = out_empty[:]
    output_row[labels.index(docs_y[x])] = 1
```

La technique du sac de mots consiste à transformer un texte en un vecteur où chaque élément représente la présence ou l'absence d'un mot spécifique dans le texte

La bibliothèque `Google Translator` est un outil conçu pour faciliter la traduction de texte en utilisant les services de traduction fournis par Google. Elle fournit une interface simple et conviviale pour accéder aux fonctionnalités de traduction automatique de Google, permettant aux développeurs d'intégrer facilement la traduction de texte dans leurs applications.

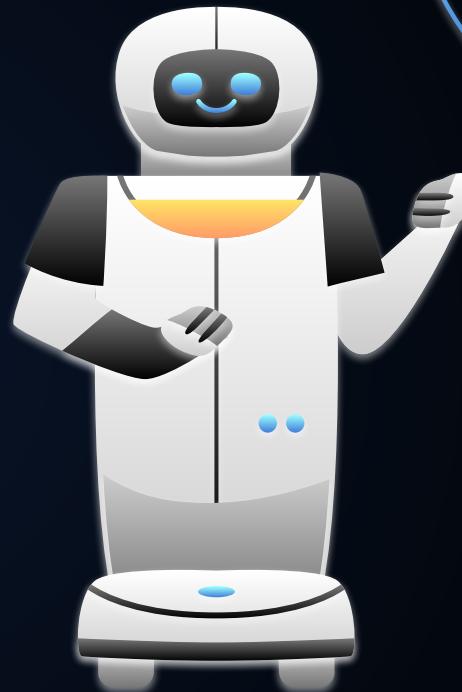
```
# Initialize Google Translator
translator = GoogleTranslator()
```

```
if "translate" in tokens:  
    # Traduction du texte si nécessaire  
    if detected_lang != 'en':  
        try:  
            # Extraire le texte à traduire (après "traduire")  
            to_translate = input_text.lower().split("translate", 1)[1]  
            translated = translator.translate(text=to_translate, source=detected_lang, target='en')  
            bot_response = f"Traduction : {translated}"
```

Ce code permet au chatbot de détecter automatiquement la langue du texte d'entrée, de le traduire en anglais s'il n'est pas déjà en anglais, et de renvoyer la traduction comme réponse

Réseaux Neuronaux

Les réseaux neuronaux sont des modèles d'apprentissage automatique inspirés par le fonctionnement du cerveau humain. Ils permettent au chatbot d'apprendre et de s'adapter en continu, améliorant ainsi sa capacité à fournir des réponses précises.



```
model = Sequential([
    Dense(8, input_shape=(len(training[0]),), activation='relu'),
    Dense(8, activation='relu'),
    Dense(len(output[0]), activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

try:
    model.load("model.h5")

```

Le modèle utilisé est un réseau de neurones artificiels construit avec Keras, composé de trois couches :

1. Couche d'entrée avec 8 neurones, utilisant la fonction d'activation 'relu' pour introduire de la non-linéarité.
2. Couche cachée également avec 8 neurones et activation 'relu'.
3. Couche de sortie avec un nombre de neurones équivalent aux classes de sortie, utilisant 'softmax' pour transformer les scores en probabilités de chaque classe.

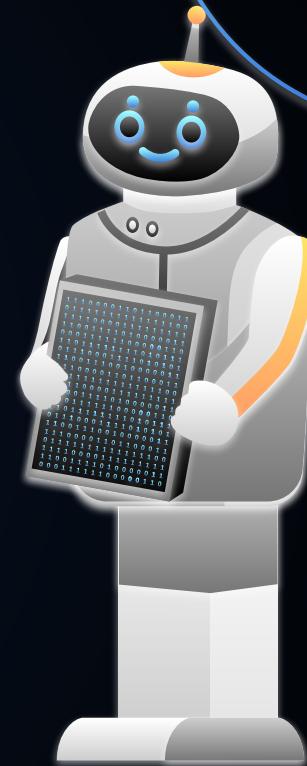
Le modèle est compilé avec :

- Optimizer 'adam' : un algorithme d'optimisation efficace pour les réseaux de neurones qui ajuste le taux d'apprentissage durant l'entraînement.
- Loss 'categorical_crossentropy' : une fonction de perte adaptée pour les classifications multi-classes, qui évalue l'écart entre les prédictions du modèle et les valeurs réelles.
- Metric 'accuracy' : pour mesurer la précision du modèle durant l'entraînement et le test.

Principe de la traduction avec LSTM

Les LSTM (Long Short-Term Memory) sont une architecture de réseau neuronal récurrent (RNN) particulièrement adaptée au traitement de séquences de données. Contrairement aux réseaux RNN traditionnels, les LSTM sont conçus pour mieux gérer les problèmes de disparition du gradient et de mémoire à court terme, ce qui les rend particulièrement adaptés à des tâches comme la traduction automatique.

Dans ce modèle, les couches LSTM sont utilisées à la fois pour l'encodage des phrases source en anglais et pour le décodage des phrases traduites en portugais. Ces LSTM capturent les informations contextuelles des phrases et génèrent des traductions précises en utilisant les mécanismes d'encodage et de décodage décrits précédemment.



Dataset Utilisée

Cette dataset contient des paires de phrases, avec une phrase en anglais dans la colonne "english" et sa traduction correspondante en portugais dans la colonne "Portuguese".

	english	Portuguese
0	Resumption of the session	Reinício da sessão
1	I declare resumed the session of the European ...	Declaro reaberta a sessão do Parlamento Europe...
2	Although, as you will have seen, the dreaded '...	Como puderam constatar, o grande "bug do ano 2...
3	You have requested a debate on this subject in...	Os senhores manifestaram o desejo de se proced...
4	In the meantime, I should like to observe a mi...	Entretanto, gostaria - como também me foi pedi...
...
1960403	I am not going to re-open the 'Millennium or n...	Não vou abrir novamente o debate sobre "miléni...
1960404	Adjournment of the session	Interrupção da sessão
1960405	I declare the session of the European Parliame...	Dou por interrompida a sessão do Parlamento Eu...
1960406	(The sitting was closed at 10.50 a.m.)	(A sessão é suspensa às 10H50)
1960407		
1960408 rows × 2 columns		

1-Pretraitement

suppression de la ponctuation

```
import re

def remove_punc(x):
    if isinstance(x, str):
        return re.sub('[!#?,.:;" ]', '', x)
    else:
        return x

# Assuming df is your DataFrame
df['Portuguese'] = df['Portuguese'].apply(remove_punc)
df['english'] = df['english'].apply(remove_punc)
```

Missing Values

```
# Check for missing values
print("Number of missing values in English:", df['english'].isnull().sum())
print("Number of missing values in Portuguese:", df['Portuguese'].isnull().sum())
# Drop rows with missing values
df.dropna(inplace=True)
```

2-analyse du Vocabulaire:

- Extraction des mots uniques :

Cette partie du code extrait et compte les mots uniques dans les colonnes "english" et "Portuguese" du DataFrame.

- Comptage des mots uniques :

Elle calcule le nombre total de mots uniques en anglais et en portugais , ce qui est essentiel pour comprendre la diversité du vocabulaire dans chaque langue.

```
english_words = []
Portuguese_words = []

# function to get the list of unique words
def get_label_superset(x, word_list):
    if isinstance(x, str): # Check if x is a string
        for label in x.split():
            if label not in word_list:
                word_list.append(label)

# Apply the function to 'english' and 'Portuguese' columns
df['english'].apply(lambda x: get_label_superset(x, english_words))
df['Portuguese'].apply(lambda x: get_label_superset(x, Portuguese_words))

# number of unique words in English
total_english_words = len(english_words)
print(f'Total unique words in English: {total_english_words}')

# number of unique words in Portuguese
total_Portuguese_words = len(Portuguese_words)
print(f'Total unique words in Portuguese: {total_Portuguese_words}')
```

2-analyse du Vocabulaire:

- Analyse de la fréquence des mots:

Le code extrait et compte les mots uniques dans la colonne.

Il utilise la bibliothèque “Counter” pour compter les occurrences de chaque mot.

Les résultats sont triés par fréquence d'occurrence, fournissant une vue succincte des mots les plus courants dans les données anglaises.

```
words_e = []

# Iterate over 'english' column to collect words
for i in df['english']:
    if isinstance(i, str):
        for word in i.split():
            words_e.append(word)

# Count occurrences of each word
english_words_counts = Counter(words_e)
english_words_counts
# Sort the dictionary by values
english_words_counts = sorted(english_words_counts.items(), key=operator.itemgetter(1),
english_words_counts
```

2-analyse du Vocabulaire:

- Analyse des longueurs (nb des mots) de documents :

Le code parcourt chaque document et tokenize ensuite le document en mots.

La longueur de chaque document est mesurée en comptant le nombre de mots.

le code affiche la longueur maximale de mots dans un document pour les deux langues .

```
import nltk
nltk.download('punkt')

# Maximum length (number of words) per document. We will need it later for embeddings
maxlen_english = -1
for doc in df['english']:
    if isinstance(doc, str): # Check if the value is a string
        tokens = nltk.word_tokenize(doc)
        if maxlen_english < len(tokens):
            maxlen_english = len(tokens)

print("The maximum number of words in any document = ", maxlen_english)

# Maximum length (number of words) per document. We will need it later for embeddings
maxlen_portuguese = -1
for doc in df['Portuguese']:
    if isinstance(doc, str): # Check if the value is a string
        tokens = nltk.word_tokenize(doc)
        if maxlen_portuguese < len(tokens):
            maxlen_portuguese = len(tokens)

print("The maximum number of words in any document = ", maxlen_portuguese)
```

2-analyse du Vocabulaire:

- Taille du vocabulaire total :

La taille du vocabulaire est calculée en ajoutant 1 au nombre total de mots uniques, pour prendre en compte le rembourrage.

```
# Total vocab size, since we added padding we add 1 to the total word count
english_vocab_size = total_english_words + 1
print("Complete English Vocab Size:", english_vocab_size)
# Total vocab size, since we added padding we add 1 to the total word count
Portuguese_vocab_size = total_Portuguese_words + 1
print("Complete portuguese Vocab Size:", Portuguese_vocab_size)
```

3-Tokenisation et rembourrage des données :

Le code utilise la classe “Tokenizer” de “Tensor Flow” pour convertir les documents textuels en séquences de nombres entiers.

Il filtre les valeurs non-chaînes et les remplace par des chaînes de caractères.

Il découpe les mots et créer des séquences de mots tokenisés.

Les données tokenisées sont ensuite rembourrées à une longueur maximale pour assurer que toutes les séquences ont la même longueur.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

def tokenize_and_pad(x, maxlen):
    # Filter out or replace non-string values
    filtered_x = [str(i) for i in x if isinstance(i, str)]

    # Initialize a tokenizer to tokenize the words and create sequences of tokenized words
    tokenizer = Tokenizer(char_level=False)
    tokenizer.fit_on_texts(filtered_x)
    sequences = tokenizer.texts_to_sequences(filtered_x)
    padded = pad_sequences(sequences, maxlen=maxlen, padding='post')
    return tokenizer, sequences, padded

# Tokenize and pad the data
e_tokenizer, e_sequences, e_padded = tokenize_and_pad(df['english'], maxlen_english)
p_tokenizer, p_sequences, p_padded = tokenize_and_pad(df['Portuguese'], maxlen_english)
```

4-obtenir le texte à partir des variables rembourrées ::

- Transforme les données rembourrées en texte lisible.
- Utilise un dictionnaire inverse pour retrouver les mots à partir de leurs identifiants.
- Assure que le remboursement est ignoré en initialisant le premier élément du dictionnaire à une chaîne vide.
- Retourne le texte reconstruit à partir des données rembourrées.

```
# function to obtain the text from padded variables
def pad_to_text(padded, tokenizer):

    id_to_word = {id: word for word, id in tokenizer.word_index.items()}
    id_to_word[0] = ''

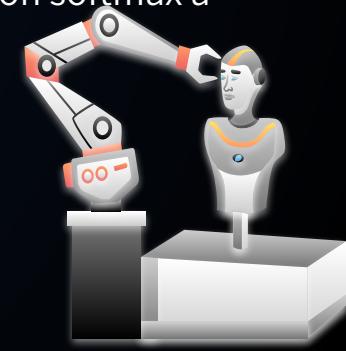
    return ' '.join([id_to_word[j] for j in padded])
pad_to_text(p_padded[0], p_tokenizer)
```

5- Définition et compilation du modèle :

Le code utilise Tensor Flow et Keras pour définir un modèle de réseau neuronal séquentiel.

Les couches suivantes sont ajoutées au modèle :

- Une couche `Embedding` : convertit les identifiants de mots en vecteurs denses.
- Une couche LSTM pour l'encodeur: capture les informations contextuelles des phrases en anglais.
- Une couche de répétition (`RepeatVector`) pour le décodeur: répète la représentation encodée pour chaque pas de temps du décodage.
- Une autre couche LSTM pour le décodeur: génère les traductions en portugais.
- Une couche `TimeDistributed` : applique une couche dense avec une fonction d'activation softmax à chaque pas de temps du décodage.
- Le modèle est compilé avec l'optimiseur Adam et la fonction de perte `sparse_categorical_crossentropy` .
- Enfin, une entrée de données factice est utilisée pour construire le modèle, et le résumé du modèle est imprimé pour afficher son architecture.



```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Embedding, LSTM, RepeatVector, TimeDistributed

# Define the model
model = tf.keras.Sequential()

# Add layers to the model
model.add(Embedding(english_vocab_size, 256, input_length=maxlen_english, mask_zero=True)) # embedding layer
model.add(LSTM(256)) # encoder
model.add(RepeatVector(maxlen_portuguese)) # repeatvector for decoder
model.add(LSTM(256, return_sequences=True)) # decoder
model.add(TimeDistributed(Dense(Portuguese_vocab_size, activation='softmax'))) # output layer

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Generate some dummy input data for building the model
dummy_input = tf.ones((1, maxlen_english)) # Assuming maxlen_english is the maximum length of English sequences

# Build the model by calling it on the dummy input
_ = model(dummy_input)

# Now, you can print the summary
model.summary()
```

6-Entraînement du modèle

```
# Train test split
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(e_padded, p_padded, test_size = 0.1)
```

```
# save the model
model.save("weights.h5")
```

```
model.fit(x_train, y_train, batch_size=256, validation_split=0.1, epochs=10)
```

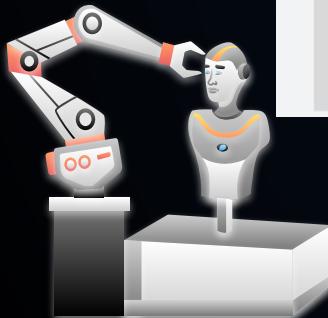
7- prédiction :

La Fonction utilise le modèle entraîné pour faire des prédictions sur les données d'entrée.

En utilisant le tokenizer de la langue cible, les prédictions sont transformées en texte.

La fonction retourne le texte prédit pour les données d'entrée.

```
# function to make prediction
def prediction(x, x_tokenizer = x_tokenizer, y_tokenizer = y_tokenizer):
    predictions = model.predict(x)[0]
    id_to_word = {id: word for word, id in y_tokenizer.word_index.items()}
    id_to_word[0] = ''
    return ' '.join([id_to_word[j] for j in np.argmax(predictions, 1)])
```



7- l'évaluation des prédictions :

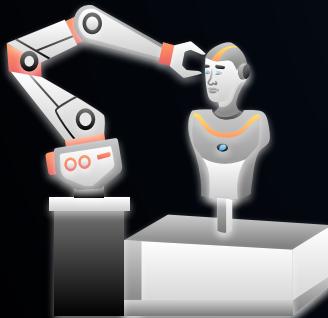
La boucle itère sur les cinq premiers exemples de données de test.

Pour chaque exemple, elle affiche le texte original en anglais et en portugais.

Ensuite, elle affiche le texte portugais prédit en utilisant la fonction prédition pour prédire la traduction à partir du texte anglais.

```
for i in range(5):

    print('Original English word - {}'.format(pad_to_text(x_test[i], x_tokenizer)))
    print('Original French word - {}'.format(pad_to_text(y_test[i], y_tokenizer)))
    print('Predicted French word - {}\\n\\n\\n'.format(prediction(x_test[i:i+1])))
```



Avantages et limites

Avantages :

Traduction en Plusieurs Langues

Accessibilité Améliorée

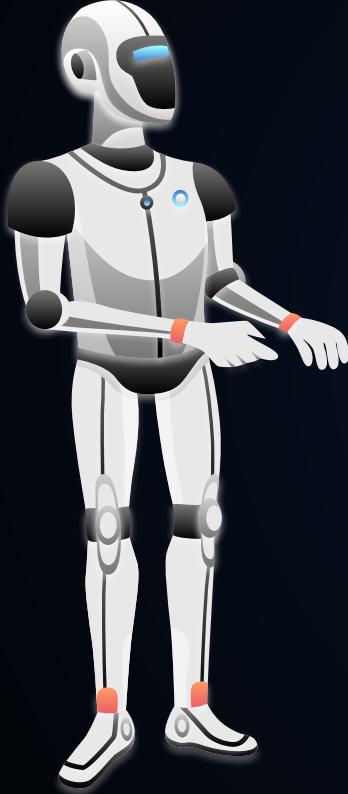
Interaction Utilisateur Améliorée

Limites :

Questions Limitées

Complexité de la Traduction

Dépendance Technologique



Conclusion

En conclusion, notre chatbot offre une expérience interactive de questions-réponses en plusieurs langues grâce à la traduction automatique.

"Merci pour votre attention."

"Obrigado pela sua atenção."

