

Rapport du projet OrientAi

2022 - 2023

Réalisé Par :

- Matouk Afaf
- El Jaouhari Mohamed
- El Amrani Ilyas
- Guerrab Mouna

Encadré Par :

Pr. Abdelghani Ghazdali

Sommaire :

- 1- Remerciement
- 2- Introduction
- 3- Objectifs
- 4- Conception
- 5- Implémentation
- 6- Expérience utilisateur
- 7- Vision
- 8- Conclusion



1-Remerciement :

Nous exprimons notre profonde gratitude envers le bon Dieu, tout puissant, pour nous avoir donné la force de persévérer et l'audace de surmonter toutes les difficultés. Nous souhaitons également adresser nos remerciements les plus sincères à Monsieur KHALFI Hamza pour ses conseils précieux tout au long du projet. Sa contribution et ses orientations avisées ont grandement enrichi notre travail et ont été d'une importance capitale pour notre succès.

En particulier, nous tenons à exprimer notre sincère gratitude envers notre cher encadrant, Monsieur GHAZDALI Abdelghani. Sa disponibilité constante et son écoute attentive tout au long de la réalisation de ce projet ont été d'une valeur inestimable. Sa guidance et son soutien indéfectibles ont joué un rôle essentiel dans notre réussite. Nous lui sommes profondément reconnaissants pour son expertise, son dévouement et sa bienveillance.



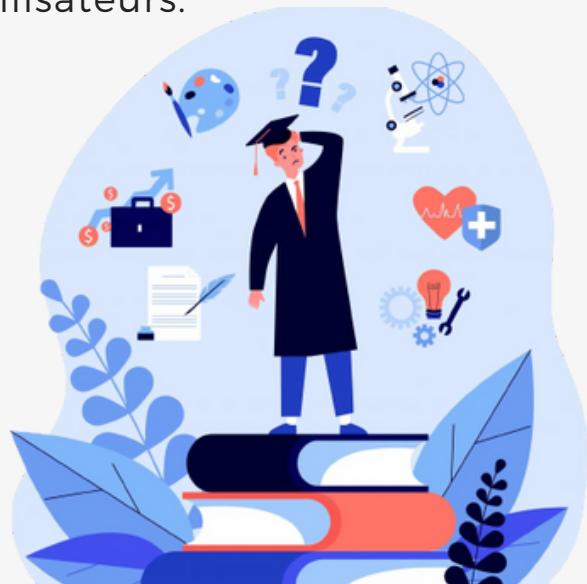
2-Introduction:

a. Problématique :

De nombreux étudiants au Maroc ont du mal à décider quelle spécialisation d'ingénierie étudier après avoir obtenu leur baccalauréat ou même après avoir terminé les classes préparatoires. Les étudiants trouvent souvent difficile de choisir une carrière parce que les écoles d'ingénierie offrent une grande variété de spécialités.

Certains étudiants peuvent ne pas avoir une compréhension claire de leurs aspirations professionnelles ou de leurs centres d'intérêt, ce qui rend difficile pour eux de sélectionner une spécialité d'ingénierie qui complète leurs intérêts et compétences. Cela ajoute à la difficulté d'orientation.

De plus, certains étudiants pourraient ne pas avoir accès à la bonne orientation ou au conseil pour les aider à prendre des décisions. Pour résoudre ce problème, nous proposons le chat-bot IMA, qui va permettre de répondre aux différentes questions et même faire des suggestions basées sur les réponses des utilisateurs.

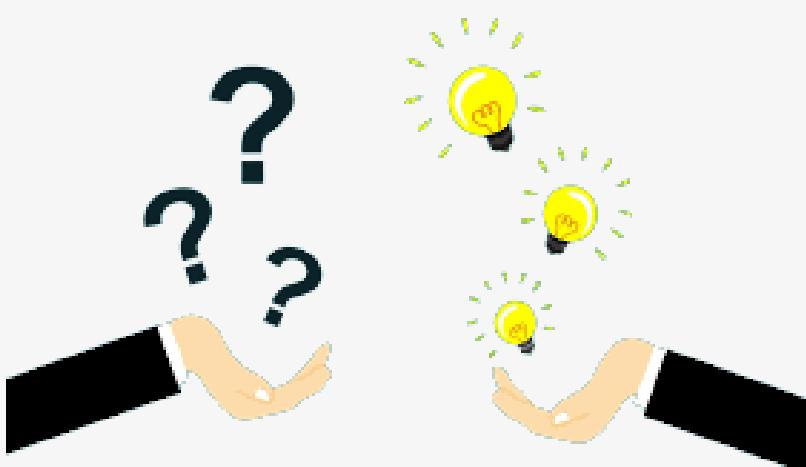


b. Définition du projet :

IMA : est un programme informatique qui utilise l'intelligence artificielle et NLP (Naturel langage processing) son but est d'aider les étudiants à découvrir leur choix d'orientation éducatifs.

Notre projet est conçu pour interagir avec les étudiants en ligne, Les poser des questions sur leurs intérêts, leurs compétences et leurs objectifs, ainsi Les fournir des informations sur les carrières ou les programmes d'étude qui pourraient les convenir

Aussi il peut également fournir des informations sur notre établissement, les programmes d'étude et les différentes possibilités pour chaque étudiant.



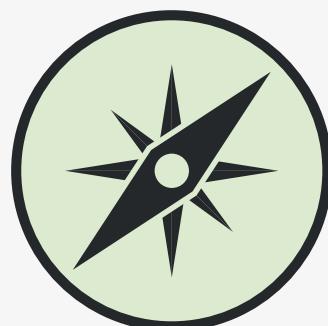
c. Périmètre :

Notre projet est destiné aux :

- Bacheliers.
- Étudiants des classes préparatoires intégrées ou extérieures participants au CNC.
- Étudiant de Formation Continue (Master).

Les cas suivants seront répondu par un refus :

- Questions sur d'autres écoles.
- Questions sur des filières non-existantes sur notre école.
- Questions sur d'autres sujets.



3-Objectifs :

1- Répondre aux questions relatives à l'école et aux différentes filières offertes par cette dernière .

2- Aider les étudiants de formation initiales (API2/ Classes préparatoires ou autre) à choisir la filière convenable pour eux en évaluant leur réponses à certaines questions, ou par le calcul de la moyenne général de chaque filières à partir de leurs notes.

3- Faciliter l'accès à l'information aux étudiants de formation continue et de présenter les programmes masters offertes par l'école en répondant à leurs questions.

4 - Conception :

4.a Chatbot IMA :

Analyseurs lexicaux :

Un tokenizer est un outil ou un algorithme utilisé dans le traitement du langage naturel (NLP) pour décomposer le texte en jetons ou en unités individuelles. Les jetons sont généralement des unités de mots, de phrases ou de sous-mots, selon la granularité du processus de tokenisation.

Il permet de transformer des données textuelles non structurées en données structurées pouvant être traitées et analysées plus efficacement.

Pour notre projet, nous avons utilisé 2 analyseurs différent qui présentent respectivement les questions et les réponses.

4 - Conception :

4.a Chatbot IMA :

Représentation vectorielle des mots :

Glove (Global Vectors for Word Representation) est un algorithme d'apprentissage non supervisé utilisé pour créer des représentations vectorielles denses (également appelées incorporations de mots) de mots dans un corpus. Le "300D" dans GloVe300D fait référence à la dimension de l'ancre de mot, et "300" indique que chaque mot est représenté par un vecteur de 300 nombres réels.

Ces ancrés de mots capturent les relations sémantiques et syntaxiques entre les mots sur la base des statistiques de cooccurrence des mots dans le corpus. Les mots de sens ou d'usage similaires ont tendance à avoir des représentations vectorielles similaires dans l'espace intégral.

4 - Conception :

4.a Chatbot IMA :

Architecture :

Notre chatbot a une architecture Seq2Seq (Sequence-to-Sequence). Ce type d'architecture est composée de 2 composantes : Encodeur & Décodeur.

L'encodeur traite la séquence d'entrée et la comprime en une représentation de longueur fixe appelée vecteur de contexte. Ce vecteur capture des informations importantes à partir de la séquence d'entrée.

Le décodeur prend un vecteur de contexte en entrée et produit une séquence de sortie pas à pas. A chaque étape, l'élément suivant de la séquence est prédict sur la base des éléments générés précédemment.

4 - Conception :

4.a Chatbot IMA :

----- API :

Notre API joue un rôle crucial de recevoir les requêtes des étudiants pour gérer ces interactions avec notre chatbot n'oublians pas que les questions des étudiants serons récupérées par la méthode POST , ces questions là seront traité par notre modèle crée pour analyser les requêtes et extraire les informations qui aident au final de générer et de renvoyer les réponses convenables et aussi de gérer l'état de la conversation pour fournir une expérience de notre chatbot interactive et fluide .



4 - Conception :

4.b QSTModel :

i. Idée de base :

Les formulaires constituent la forme de base pour récolter des données, l'analyse aux réponses et la mesure de similarité peut révéler les centres d'intérêt communs entre les étudiants ingénieurs actuelles dans les divers filières et l'utilisateur du site web.

Nous avons créé un modèle en apprentissage semi-supervisé pour pouvoir calculer le niveau de similarité entre un étudiant (remplissant le formulaire) et les étudiants présents dans la base de données ,qui sont des étudiants en cycle d'ingénieur, satisfait de leur choix, ce modèle retourne les probabilité de compatibilité de cet étudiant avec les cinq filières de l'école.

Nous avons pu récolté une petite base de données qui comprend les réponses aux questions et la filière choisie.

4 - Conception :

4.b QSTModel :

ii. Le modèle :

Notre modèle est constitué de deux parties, qui sont utilisés suivant le cas rencontré lors de la classification de notre étudiant sur les 5 classes (filières) probables, en retournant les probabilités d'appartenance à chaque classe.

Le premier modèle est un classificateur KNN (K Nearest neighbors, ou Les K voisins les plus proches) avec K par défaut égale à cinq, il est entraîné sur notre dataset, et sera utilisé dans un cas spéciale qu'on détaillera par la suite.

Le deuxième modèle est un classificateur KMeans qui fait du clustering, sur toute la base de données (y compris le nouveau étudiant) et par la suite essaye d'établir une majorité sur chaque cluster d'étudiants, si cette dernière ne peut pas être établie par raison d'égalité de nombre d'étudiants des classes présentes, nous utilisons le modèle KNN pour prédire les probabilité d'appartenance aux classes.

4 - Conception :

4.b QSTModel :

iii. Les Données :

Initialement les données sont composés de 25 colonne, y compris 20 des réponses aux questions, le nom, prénom, email, filière et date de récupération des données.

Notre modèle prend comme variables de décision les réponses aux questions de 1 à 20 et essaye de prédire une des filières (codées de 1 à 5), pour faire cela, nous aurons besoin de transformer nos données, pour une instance (une requête) qui regroupe les données de l'utilisateur nous utiliserons le même algorithme pour transformer nos données, l'algorithme est constitué des étapes suivantes :

1. Supprimer les colonnes facultatifs (nom, email ..).
2. Encoder les réponses selon un dictionnaire qui associe un poids à chaque réponse.
3. Encoder la filière (si elle est présente dans les données de la requête).
4. Renvoyer le résultat.

Vers la fin de cet algorithme nous aurons 21 colonnes (20 pour les questions 'Qi' et une pour la filière 'major'). Ces données seront divisées par la suite en X (matrice des données d'entrée des modèles) et y (les filières respectives de chaque enregistrement.)

4 - Conception :

4.b QSTModel :

iv. L'API :

Pour notre API, nous implémenterons une page pour le formulaire (pour la récupération des données), et une autre pour l'affichage. Les données seront récupérées par méthode POST et seront traité par le modèle qui renvoie les probabilités d'appartenance aux différentes filières sous forme d'un dictionnaire, dont les clés sont les noms des filières et les valeurs sont les pourcentage de probabilités d'appartenance. Ces pourcentages sont utilisés pour déterminer la classe de probabilité maximale, ce qui déterminera la description et l'image à transmettre avec nos probabilités à une template HTML pour afficher les résultats.



4 - Conception :

4.c NoteModel :

i. Algorithme :

Notre Algorithme suggéré la filière convenable à l'étudiant on se basant sur ses notes.

Cette suggestion se fait par le calcul la moyenne General de chaque Filière .

Cette moyenne est basé sur les notes de l'étudiant pour les 24 modules (modules des 4 semestres du cycle préparatoire), ainsi les coefficients correspondant à chaque module pour les 5 filières (IID, GI, GPEE, GE, IRIC)

En fin, notre programme classe les filières selon leurs moyenne calculé.

4 - Conception :

4.c NoteModel :

iv. L'API :

Nous implémenterons une page pour le formulaire où chaque étudiant peut faire entrer ces notes pour chaque module , ces notes là seront récupérées par la méthode POST et seront traiter par le modèle qui renvoie la moyenne générale pour chaque filière afin que l'étudiant sait la filière convenable d'après ses notes ses moyennes là seront afficher dans une page HTML sous formes d'un tableau.



5 - Implémentation :

5.a Chatbot IMA :

Etape 1 :

La 1ère étape consiste à lire les données CSV en les transformants en une DataFrame contenant des pairs Question-Réponse. Ensuite, on applique un pipeline pour gérer les erreurs communes.

```
df = pd.read_csv("./chatbot_data.csv",sep="|")

for count,que in enumerate(df["Questions"]):
    df["Questions"][count] = str( df["Questions"][count])
for count,que in enumerate(df["Answers"]):
    df["Answers"][count] = str( df["Answers"][count])

def pipeline(df):
    for count in enumerate(df["Questions"]):
        df.iloc[count,0] = re.sub("[^a-zA-Z]+", " ", df.iloc[count,0])
        df.iloc[count,0] = df.iloc[count,0].lower()
        df.iloc[count,0] = re.sub("I'm", "I am", df.iloc[count,0])
        df.iloc[count,0] = re.sub("he's", "he is", df.iloc[count,0])
        df.iloc[count,0] = re.sub("she's", "she is", df.iloc[count,0])
        df.iloc[count,0] = re.sub("it's", "it is", df.iloc[count,0])
        df.iloc[count,0] = re.sub("that's", "that is", df.iloc[count,0])
        df.iloc[count,0] = re.sub("what's", "that is", df.iloc[count,0])
        df.iloc[count,0] = re.sub("where's", "where is", df.iloc[count,0])
        df.iloc[count,0] = re.sub("how's", "how is", df.iloc[count,0])
        df.iloc[count,0] = re.sub("\'ll", " will", df.iloc[count,0])
        df.iloc[count,0] = re.sub("\'ve", " have", df.iloc[count,0])
        df.iloc[count,0] = re.sub("\'re", " are", df.iloc[count,0])
        df.iloc[count,0] = re.sub("\'d", " would", df.iloc[count,0])
        df.iloc[count,0] = re.sub("\'re", " are", df.iloc[count,0])
        df.iloc[count,0] = re.sub("won't", "will not", df.iloc[count,0])
        df.iloc[count,0] = re.sub("can't", "cannot", df.iloc[count,0])
        df.iloc[count,0] = re.sub("n't", " not", df.iloc[count,0])
        df.iloc[count,0] = re.sub("til", "until", df.iloc[count,0])
        df.iloc[count,0] = re.sub("\?", " ? ", df.iloc[count,0])
        df.iloc[count,0] = " ".join( df.iloc[count,0].split())
    df["Questions"] = df["Questions"].apply(lambda text: lemmatize_words(text))
```

```

for count in enumerate(df["Answers"]):
    df.iloc[count,1] = re.sub(r"[^a-zA-Z0-9:-]+", " ", df.iloc[count,1])
    df.iloc[count,1] = df.iloc[count,1].lower()
    df.iloc[count,1] = re.sub(r"I'm", "I am", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"he's", "he is", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"she's", "she is", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"it's", "it is", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"that's", "that is", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"What's", "that is", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"where's", "where is", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"\?", " ? ", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"how's", "how is", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"\'ll", " will", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"\'ve", " have", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"\'re", " are", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"\'d", " would", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"\'re", " are", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"won't", "will not", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"can't", "cannot", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"n't", " not", df.iloc[count,1])
    df.iloc[count,1] = re.sub(r"'til", "until", df.iloc[count,1])
    df.iloc[count,1] = " ".join( df.iloc[count,1].split())
df["Answers"] = df["Answers"].apply(lambda text: lemmatize_words(text))
return df

```

Etape 2 :

La 2ème étape est pour d'une part, normaliser les mots en utilisant une présentation ASCII simplifiée ainsi qu'ajouter d'autre part 2 mots : <start> , <end> qui indiquent respectivement le début et la fin d'une page.

```

def unicode_to_ascii(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn')

def preprocess_sentence(w):
    w = unicode_to_ascii(w.lower().strip())
    w = '<start> ' + w + ' <end>'
    return w

```

Etape 3 :

Pour cette 3ème étape, le but est de créer 2 Analyseurs lexicaux, le premier pour les Questions, et le 2ème pour les réponses. cela est possible à l'aide du code suivant :

```
def tokenize(lang):

    tokenizer = tf.keras.preprocessing.text.Tokenizer(filters='', lower=True, oov_token=<UNK>)
    tokenizer.fit_on_texts(lang)
    VOCAB_SIZE = len(tokenizer.word_index) +1

    lang_tokenizer = tokenizer

    tensor = lang_tokenizer.texts_to_sequences(lang)

    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
    padding='post', truncating="post")

    return tensor, lang_tokenizer,VOCAB_SIZE
```

Etape 4 :

A ce niveau, on passe à télécharger le document Glove de dimension 300 et puis mettre à chaque mot dans un analyseur lexicaux la présentation vectorielle correspondante.

```
embeddings_index = {}
with open("./glove.6B.300d.txt", 'r',encoding='utf-8') as f:

    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

```

# pour les questions
embedding_matrix_que = np.zeros((vocab_quest, 300))
for word, i in inp_lang.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_que[i] = embedding_vector

# pour les réponses
embedding_matrix_ansew = np.zeros((vocab_ansew, 300))
for word, i in targ_lang.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix_ansew[i] = embedding_vector

```

Etape 5 :

Dans cette étape, on aimera expliquer les classes suivantes :

Encodeur :

Notre classe d'encodeur contient le constructeur suivant :

```

def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz=0):
    super(Encoder, self).__init__()
    self.batch_sz = batch_sz
    self.enc_units = enc_units
    self.embedding = tf.keras.layers.Embedding(vocab_size,
                                                embedding_dim,
                                                weights=[embedding_matrix_que],
                                                trainable=False, mask_zero=True)

    self.gru = tf.keras.layers.GRU(self.enc_units,
                                  return_sequences=True,
                                  return_state=True,
                                  recurrent_initializer='glorot_uniform')

```

- Nombre de donnée à entrée par epoch.
- nombre de neurones : ce-ci sera utiliser dans la couche GRU.
- Couche d'emebedding : notons que les poids de cette couche sont obtenu par la présentation de glove.
- Couche de composante GRU.

Tous les composantes sont instanciée à l'aide de leur implémentation prédéfinie dans Tensorflow.

Ensuite, la méthode **call** est utilisée pour l'entraînement/exécution de l'encodeur en passant l'entrée par la couche d'embedding à fin d'obtenir la présentation vectorielle et puis entrer cette présentation dans la couche de GRU pour obtenir la **séquence de sortie** et **l'état finale** qui sera considérée comme un vecteur de contenu.

```
def call(self, x, hidden):
    x = self.embedding(x)
    output, state = self.gru(x, initial_state = hidden)
    return output, state
```

Enfin, nous introduisons la fonction **initialize_hidden_state** qui **initialise les poids** d'encodeur dans chaque epoch d'entraînement.

```
def initialize_hidden_state(self):
    return tf.zeros((self.batch_sz, self.enc_units))
```

Mécanisme d'attention de Bahdanau:

Le mécanisme d'attention de Bahdanau aide le modèle à se concentrer sur différentes parties de la séquence d'entrée à mesure que chaque élément de la séquence de sortie est généré. Pour ce faire, des poids sont attribués à chaque élément d'entrée et une somme pondérée des éléments d'entrée est calculée sur la base de ces poids.

Le mécanisme s'exécute comme suivant :

- En entrée, on a les états cachées d'encodeur et un état cachée du décodeur.
- On combine l'état de chaque encodeur avec l'état du décodeur pour obtenir un score à l'aide d'une couche de neurones simples. ce score sera envoyée à la fonction softmax pour obtenir les **poids d'attentions**.
- On multiplie chaque état cachée d'encodeur par le poids d'attention correspondant. Ensuite, on somme tous les états après la multiplication. la somme de ces états est appelée le **vecteur contenu**. ce vecteur représente l'information générée par la séquence d'entrée.
- le vecteur contenu et l'état cachée de décodeur seront concaténée dont le vecteur résultant sera utilisée pour prédire l'élément prochain de la séquence de sortie.

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
        query_with_time_axis = tf.expand_dims(query, 1)

        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        attention_weights = tf.nn.softmax(score, axis=1)

        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

Décodeur :

On commence par le constructeur de notre classe :

```
def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
    super(Decoder, self).__init__()
    self.batch_sz = batch_sz
    self.dec_units = dec_units
    self.embedding = tf.keras.layers.Embedding(vocab_size,
                                                embedding_dim,
                                                weights=[embedding_matrix_anse],
                                                trainable=False, mask_zero=True)
    self.gru = tf.keras.layers.GRU(self.dec_units,
                                  return_sequences=True,
                                  return_state=True,
                                  recurrent_initializer='glorot_uniform')
    self.fc = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.dec_units)
```

Le constructeur est similaire à celui d'encodeur, la différence est au niveau des poids puisque chaque matrice des poids est appropriée à un **tokenizer** (celui des réponses dans ce cas d'encodeur) ainsi que l'ajout du mécanisme d'attention pour aider la prédiction de l'encodeur.

Ensuite, la fonction call de notre modèle commence par utilisée le mécanisme d'attention. Après obtenir le vecteur contenu, on le concatène avec la représentation glove et on le passe dans la couche GRU et enfin la sortie de cette couche passe par une couche de neurone simples dont le nombre est égale à la taille du dictionnaire (tokenizer des réponses).

```
def call(self, x, hidden, enc_output):

    context_vector, attention_weights = self.attention(hidden, enc_output)

    x = self.embedding(x)

    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    output, state = self.gru(x)

    output = tf.reshape(output, (-1, output.shape[2]))

    x = self.fc(output)

    return x, state, attention_weights
```

Etape 6 :

Pour cette étape, on incetancie l'encodeur, le mécanisme d'attention et le décodeur et on initialise leur poids

```
# longeur maximale des séquences entrées
BUFFER_SIZE = len(input_tensor)

BATCH_SIZE = 64

# nombre de pas par epoch
steps_per_epoch = len(input_tensor)//BATCH_SIZE

embedding_dim = 300

units = 512
```

```
encoder = Encoder(vocab_quest, embedding_dim, units, BATCH_SIZE)

sample_hidden = encoder.initialize_hidden_state()

sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
```

```
attention_layer = BahdanauAttention(128)  
attention_result, attention_weights = attention_layer(sample_hidden, sample_output)
```

Etape 8 :

Dans cette étape, nous concentrons sur la partie entraînement.

On commencera par définir l'optimizer (Adam dans notre cas), la fonction loss (Sparse CategoricalCrossEntropy pour la classification multi-class) et une fonction python responsable sur l'évaluation de l'erreur au cours de l'entraînement.

```
optimizer = tf.keras.optimizers.Adam()

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Ensuite, on passe à expliquer notre fonction qui permet d'appliquer l'entraînement pour 1 epoch.

- Le décorateur @tf.function a pour but de rendre la fonction performante en la transformant en graphe.
- Gradient_tape permet d'automatiser la différentiation suivant les variables d'entraînement.

- On entraîne l'encodeur et on prend sa sortie (état finale) à fin de l'insérer comme initialisation d'état pour le décodeur. Ensuite, puisque nous travaillons sur des batch (groupement des données) et non un seul pair Question - Réponse, on crées le nombre de réponses dans le groupement des données et on les ajoute <start> au début.
- On calcule la somme des pertes (loss) pour chaque paire Q&R après le calcul des pertes individuelles entre les valeurs réelles et celles prédictives.
- En utilisant la technique de "teacher forcing" qui se base à prendre la vraie valeur comme entrée pour l'entraînement.
- En dernier, on enregistre les variables entraînées, calculer le gradient respectivement aux variables et puis on optimise les variables entraînées.

```

@tf.function
def train_step(inp, targ, enc_hidden):
    loss = 0

    with tf.GradientTape() as tape:

        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([targ_lang.word_index['<start>']] * BATCH_SIZE, 1)

        for t in range(1, targ.shape[1]):

            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)

            loss += loss_function(targ[:, t], predictions)

            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = (loss / int(targ.shape[1]))

    variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, variables)

    optimizer.apply_gradients(zip(gradients, variables))

    return batch_loss

```

Pour lancer l'entraînement, on utilise le code suivant en précisant le nombre d'epochs voulu :

```
EPOCHS = 450

for epoch in range(1, EPOCHS + 1):
    enc_hidden = encoder.initialize_hidden_state()
    total_loss = 0
    print(f"Starting epoch {epoch}...")
    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
        batch_loss = train_step(inp, targ, enc_hidden)
        total_loss += batch_loss
    print('Ending Epoch:{:3d} Loss:{:.4f}'.format(epoch, total_loss / steps_per_epoch))
print("Ended")
```

Etape 9 :

A ce stade, on enregistre les poids d'encodeur, décodeur, ainsi que nous enregistrons les **tokenizers** à l'aide du module **pickle**.

```
encoder.save_weights("./weights/encoder/encoder_512_ep450_bs64.h5")
decoder.save_weights("./weights/decoder/decoder_512_ep450_bs64.h5")

with open('./tokinzers/tokenizers_ques.pickle', 'wb') as handle:
    pickle.dump(inp_lang, handle, protocol=pickle.HIGHEST_PROTOCOL)

with open('./tokinzers/tokenizers_answ.pickle', 'wb') as handle:
    pickle.dump(targ_lang, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Etape 10 :

Cette étape est pour la communication avec IMA. Une fonction dédiée est créée à fin de pouvoir d'utiliser notre chatbot.

Comme début, on commence par passer notre entrée dans le pipeline et puis on ajoute **<start>** & **<end>** pour la séquence sortie du pipeline. Ensuite, on crée une liste contenant les entiers ayant une liaison respective avec les mots présents dans la séquence.

Pour confirmer que la séquence a la taille requis pour entrer dans l'encodeur, on utilise **pad_sequences** pour cette tache.

```
def remove_tags(sentence):
    return sentence.split("<start>")[-1].split("<end>")[0]

def evaluate(sentence,encoder,decoder):
    sentence = pipeline(sentence)
    sentence = preprocess_sentence(sentence)
    sentence.strip()
    inputs = [tokenizer_quest.word_index[i] for i in sentence.split(" ")]
    inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                          maxlen=max_length_inp,
                                                          padding='post')
    inputs = tf.convert_to_tensor(inputs)
```

Le code que nous allons montrer ensuite a comme but d'obtenir le vecteur contenu par l'encodeur, puis initialiser l'état finale de décodeur par l'état finale d'encodeur

```
result = ''  
  
hidden = [tf.zeros((1, self.units))]  
enc_out, enc_hidden = self.encoder(inputs, hidden)  
  
dec_hidden = enc_hidden  
dec_input = tf.expand_dims([self.tokenizer_anw.word_index['<start>']], 0)
```

On passe à la partie de prédiction où on introduira une boucle qui utilise le décodeur pour qui implémentera les poids d'attentions et puis prédire le mot de la séquence de sortie sous format de vecteur dans chaque itération (sans oublier <start> en début). on prend l'entier maximale qui représente le mot dans l'analyseur lexicaux à l'aide de la fonction **argmax**.

```
for t in range(self.max_length_targ):  
    predictions, dec_hidden, attention_weights = self.decoder(dec_input,  
                                                               dec_hidden,  
                                                               enc_out)  
  
    attention_weights = tf.reshape(attention_weights, (-1, ))  
  
    predicted_id = tf.argmax(predictions[0]).numpy()  
  
    result += self.tokenizer_anw.index_word[predicted_id] + ' '  
  
    if self.tokenizer_anw.index_word[predicted_id] == '<end>':  
        return self.remove_tags(result), self.remove_tags(sentence)  
  
    dec_input = tf.expand_dims([predicted_id], 0)  
  
return self.remove_tags(result), self.remove_tags(sentence)
```

Cette boucle nous permettra de prédire mot par mot jusqu'à trouver **<end>** où on doit s'arrêter.

Etape 11 : API

Au début on a créé un projet django pour qu'on peut faire la liaison et une interaction entre notre modèle du chatbot et les interfaces créer et poser dans le dossier 'template'. Puis on a notre application 'myapii', où on trouve les fichiers apps.py et views.py chaque'un d'eux à son rôle .

settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'rest_framework',
    'myapii',
```

Où on a fait ajouter le nom de l'application pour indiquer que l'application fait partie du projet, configurer des paramètres spécifiques et permettre une meilleure organisation, interopérabilité et réutilisabilité du code

urls.py

```
from django.contrib import admin
from django.urls import path
from myapii import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('algo', views.index),
    path('myAPI', views.chatbot_api_view, name='chatbot_api')
```

Ce fichier est responsable sur la définition des URL, de leur association aux vues appropriées (dans le fichier views.py), et de la gestion des paramètres d'URL Il joue aussi un rôle clé dans le routage des requêtes vers les bonnes vues de votre application

Views.py

```
from django.shortcuts import render
from .apps import MyapiConfig
from django.template import loader
from django.http import JsonResponse, HttpResponseRedirect
import datetime
from django.apps import apps
from .chat_med.chatbot import chatbot,predict
# Cette vue sera responsable de la communication entre l'API et votre modèle de chatbot.
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def chatbot_api_view(request):
    if request.method == 'POST':
        data = request.POST.get('msg') # Récupérer le message envoyé par le chatbot
        # Traiter le message et préparer la réponse
        app_config = apps.get_app_config('myapi')
        chatbot = app_config.chatbot
        print(f"data={data}")
        response = {
            'ques': data, # Echo the chatbot's question
            'res': predict(chatbot=chatbot_model,sentence=data), # Replace with your chatbot's actual response
            'time': datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") # Current date and time
        }
        return JsonResponse(response)
```

Ce fichier contient les vues qui reçoivent les requêtes HTTP, traitent les données, effectuent des opérations métier et renvoient des réponses HTTP , et il permet aussi de définir comment les interactions avec les utilisateurs sont gérées

'chatbot_api_view' joue le rôle d'une vue pour notre API de chatbot , ou :

-il y a la vérification si la méthode de la requête est post , puis il y a la récupération du message envoyé par le chatbot n'oublions pas son traitement puis la préparation de la réponse (la fonction predict est supposée de renvoyer la réponse du chatbot en fonction du message). Au final on aura une réponse sous forme de Json en utilisant la fonction JsonResponse. Cette dernière sera consommée par le chatbot pour afficher la réponse appropriée.

```
def index(request):
    # index c'est le nom de la page html dans le dossier views
    template = loader.get_template('index.html')
    return HttpResponseRedirect(template.render({}, request))
```

Apps.py

Index ici joue le rôle de la vue pour la page 'index.html' , ou au début on remarque que cette fonction reçoit un objet qui représente la requête http entrante de l'utilisateur et renvoie une instance de la classe HttpResponseRedirect qui encapsule le contenu du 'index.html' généré à partir du template rendu. Cela permet de renvoyer la page HTML en réponse à la requête.

```
from django.apps import AppConfig
from .chat_med.chatbot import chatbot

class MyapiiConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'myapii'
    chatbot_model=chatbot()
    chatbot_model.launch()
```

Ici ce code définit une configuration d'application (**MyapiiConfig**) pour notre application Django (**myapii**) et au début on fait importer la classe AppConfig du module django.apps, cette classe est utilisée pour configurer les applications Django, puis on importe la classe chatbot depuis le module chatbot situé dans le répertoire `chat_med` de notre application. Et on fait la création d'une instance de la classe chatbot et on fait appelle de la méthode `launch` qui est supposée de lancer le chatbot et effectuer toute initialisation nécessaire.

Index.html

Dans notre page html 'index.html' , on a la présentation de l'interface de notre chatbot

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Chat Bot IMA</title>
    <link rel="stylesheet" href="/static/css/bootstrap.min.css">
    <link rel="stylesheet" href="/static/css/css/common-page.css">
    <link rel="stylesheet" href="/static/css/font-awesome-4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="/static/css/toastr.css">
    <script src="/static/script/jquery-1.10.2.js"></script>
    <script src="/static/script/bootstrap.min.js"></script>
    <script src="/static/script/toastr.js"></script>
</head>

<body>
    <section>
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <div class="row cnt">
                        <div class="col-md-12">
                            <h3><i class="fa fa-commenting"></i>Welcome to IMA</h3>
                        </div>
                        <div class="col-md-12">
                            <div class="row" id="MessageContent">
                            </div>
                        </div>
                        <div class="col-md-12">
                            <hr>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </section>
</body>
```

Le rôle du script Javascript ci-dessous est de faire la liaison entre mon interface et mon modèle du chatbot, gérer l'envoi asynchrone d'un formulaire de chat et de mettre à jour le contenu du chat en fonction de la réponse reçue.

```
<script>
    // Prevent the default form submission behavior
    document.getElementById('chatForm').addEventListener('submit', function(event) {
        event.preventDefault(); // Prevent the form from submitting normally

        // Get the form data
        var formData = new FormData(event.target);

        // Send the form data asynchronously using AJAX
        fetch(event.target.action, {
            method: 'POST',
            body: formData
        })
        .then(function(response) {
            return response.json(); // Parse the JSON response
        })
        .then(function(res) {
            drawMessage(res); // Handle the response
        })
        .catch(function(error) {
            toastr.error("Something went wrong"); // Display an error message
        });
    });
</script>
```

```
// Function to handle the response and update the message content
function drawMessage(res) {
    var msgContent = document.getElementById('MessageContent');
    var message = "<div class='col-md-6 owner-message'><p>" + res.ques + "</p><p class='time'>" +
    message += "<div class='col-md-6 guest-message'><p>" + res.res + "</p><p class='time'>" +
    msgContent.innerHTML += message;

    // Clear the text area
    var msgText = document.getElementById('Message');
    msgText.value = '';
}
</script>
```

Au début, on a eu l'attaché un gestionnaire d'événement à l'événement "submit" du formulaire ayant l'ID "chatForm". Cela permet de capturer la soumission du formulaire , puis on a la fonction 'event.preventDefault()' qui empêche le comportement de soumission par défaut du formulaire, qui recharge habituellement la page . Après cela on a la récupération et le traitement des réponses n'oublians pas la gestion des erreurs survenues lors de la requête AJAX et au final un mise à jour le contenu du chat en ajoutant les messages envoyés et reçus à la variable message par la fonction La fonction 'drawMessage(res)'

5 - Implémentation :

5.B QSTModel :

La classe QModel :

Pour implémenter le modèle décrit avant, nous nous servons de la programmation orientée objet en python, nous avons crée la classe QModel qui permet d'instancier un model semi-superviser qui sert de deux types de modèles (KNN et KMeans).

Notre modèle est basé sur le fichier CSV 'formData.csv' qui regroupe les données récupérer du questionnaire partagé sur Google Forms avec les étudiants de cycle d'ingénieur.

Pour l'initialisation (l'instanciation), nous chargeons et mettons en forme nos données, les dictionnaires d'encodage et nous chargeons le model KNN si spécifié. Nous avons détaillé ces étapes avec des méthodes, chacune prend en charge une partie.

```
#Defaults :
defaultMajors = {'IID':1,'GI':2,'GE':3,'GPEE':4,'IRIC':5}
defaultQ_answers = {'I love it':2,'I like it':1,'I am not sure':0,'Yes':1,'No':-1,
                    'I haven\'t tried it yet':0,'I am very familiar with them':2,'I know the basics':1,
                    'I have limited experience':0,'I have no experience with them':0,'I have little to no experience':0}
```

Dictionnaires d'encodage par défaut

5 - Implémentation :

5.B QSTModel :

La classe QModel :

```
def __init__(self,dataFile='./Ilyas/isIID/formData.csv',
           majorsEncoding=defaultMajors,Q_ansEncoding=defaultQ_answers,KNNmodelFile="KNNSupModel.pickle",loadKNNOnStart = False):
    ...
    dataFile : File regrouping data to use for model training.
    majorsEncoding : Dictionary to encode the majors.
    Q_ansEncoding : Dictionary to use for question answers encoding.
    KNNmodelFile : The file where to load and save the KNN (supervised) model
    LoadKNNOnStart : True to load the KNN model on the initialisation
    ...
    self.dataFile = dataFile
    self.majors = majorsEncoding
    #Creating the reverse of dictionary, keys are the numbers and values are the encoded name of the major.
    self.majorsR = {v:k for v,k in enumerate(self.majors.keys(),1)}
    self.Q_ansEncoding = Q_ansEncoding
    #Load the data
    self.loadData()

    #Loading the KNN model if specified, else configure it not created yet, nor loaded, finally consider the data not clustered yet
    self.KNNmFile = KNNmodelFile
    self.KNNCreated = False
    self.KNNLoaded = self.LoadKNN() if loadKNNOnStart else False
    self.clustered = False
```

La fonction LoadData permet de charger les données; les transformer avec la fonction preproc et séparer les colonnes des variables de décisions avec la colonne 'major', notre target.

```
#Fonctions de chargements des données :
def loadData(self):
    self.data = pd.read_csv(self.dataFile)
    self.pData = self.preproc(self.data.values)
    self.X = self.pData.drop('Major',axis=1)
    self.y = self.pData.Major
    return True
```

5 - Implémentation :

5.B QSTModel :

La classe QModel :

```
#Preprocessing Function
def preproc(self,x):
    if(type(x)==list):
        x=np.array(x)
    assert(x.size==25 or x.shape[1]==25)
    if(len(x.shape)==1):
        x=np.array([x])
    inData = pd.DataFrame(x[:,4:],columns=[ 'Major']+['Q'+str(i) for i in range(1,21)])
    inData.replace(self.majors,inplace=True)
    inData.replace(self.Q_ansEncoding,inplace=True)
    return inData
```

La fonction preproc transforme les données après la vérification de leur taille (nombre de colonnes==25 et doivent être de type numpy.array). En suite nous prenons que les colonnes de filière et des réponses des questions, et nous remplçons à l'aide dictionnaires les valeurs respectives de ces colonnes.

5 - Implémentation :

5.B QSTModel :

La classe QModel :

Notre modèle utilise un modèle KNN (supervisé) qui est contrôlé à l'aide des méthodes suivantes :

CreateSuperKNN: Crée un modèle KNN de la bibliothèque Scikit-learn, l'entraîne sur les données, enregistre ses poids et mis à jours le drapeau indicant la création de modèle KNN.

LoadKNN : Permet de charger le modèle KNN du fichier si le fichier existe.

```
#Créer et entraîner le modèle supervisé
def CreateSupKNN(self):
    self.KNNmodel = KNeighborsClassifier()
    self.KNNmodel.fit(self.X, self.y)
    pickle.dump(self.KNNmodel, open(self.KNNmFile, "wb"))
    self.KNNCreated=True
    return True

#Charger le modèle s'il existe
def LoadKNN(self):
    if.isfile(abspath(self.KNNmFile)):
        try:
            self.KNNmodel = pickle.load(open(self.KNNmFile, "rb"))
            self.KNNLoaded=True
            return True
        except:
            pass
    return False
```

5 - Implémentation :

5.B QSTModel :

La classe QModel :

Notre modèle utilise principalement un modèle d'apprentissage non supervisé (KMeans) qui est créé à chaque instanciation de modèle, et qui est utilisé en première lieu pour évaluer les probabilités des similarité de l'utilisateurs avec les étudiants d'une filière ayant répondu au questionnaire.

La fonction CreateMeans permet de créer le modèle l'entraîner et faire le clustering sur nos données.

```
#Créer et trainer le model non supervisé
def CreatekMeans(self):
    KMmodel = KMeans(5,random_state=1)
    y_clusts = KMmodel.fit_predict(self.X)
    self.clustered = True
    return y_clusts
```

Cette méthode est utilisé dans la méthode évaluate qui permet de retourner les probabiliés d'appartenance aux différentes filières pour un étudiant.

5 - Implémentation :

5.B QSTModel :

La classe QModel :

```
def Evaluate(self,x):
    x= self.preproc(x)
    self.X.add(x)
    y_clusts = self.CreateKMeans()
    y_clusts = pd.Series(y_clusts)
    y_clusts.value_counts()
    maj = self.getMajors(y_clusts,self.y,self.X)
    return maj
```

Cette dernière sert de la fonction getMajors pour calculer les probabilités, qui essaie de calculer pour chaque filière prédictive avec KMeans, de déterminer les étudiants classés par l'algorithme dans ce cluster, ensuite savoir la classe d'origine de ces étudiants et compter le nombre d'étudiants de chaque filière et déterminer la filière dominante dans la classe, lorsque une dominance est établie, nous calculons les pourcentages de présence de chaque filière, c'est le pourcentage d'appartenance de l'étudiant.

5 - Implémentation :

5.B QSTModel :

La classe QModel :

```
def getMajors(self,y,y_org,X):
    clusterMajors = []
    for clust in y.unique():
        clustEtds = y[y==clust]
        clustEtdsReel = y_org.iloc[clustEtds.index].value_counts()
        max = clustEtdsReel.max()
        #Nombre des classes dominantes
        nbCs = len(clustEtdsReel[clustEtdsReel==max].to_list())
        #On assume que la classe sera celle de la majorité
        if nbCs == 1:
            #Retourner les probabilités
            clss = clustEtdsReel.to_dict()
            total = sum(list(clss.values()))
            probas = {}
            for clse in clss.keys() :
                if(round(clss[clse]*100,3)!=0):
                    probas[self.majorsR[clse]] = round((clss[clse]/total)*100,3)
            clusterMajors[clust] = probas
        #Si la majorité n'est pas déterminée, on utilise le modèle supervisé
        else :
            if(not self.KNNLoaded):
                if(not self.LoadKNN()):
                    self.CreateSupKNN()
            Xm = X.iloc[-1,:].values.reshape(1,-1)
            probas = {}
            prbs = self.KNNmodel.predict_proba(Xm)[0,:]
            for proba in range(0,len(prbs)):
                if(round(prbs[proba]*100,3)!=0):
                    probas[self.majorsR[proba+1]] = prbs[proba]*100
            clusterMajors[clust] = probas
    return clusterMajors[clustEtds.iloc[-1]]
```

5 - Implémentation :

5.B QSTModel :

Le modèle sur Django:

Pour utiliser notre modèle, nous créons une simple page en HTML qui sert à grouper les réponses de l'utilisateur et les envoyer avec la méthode POST à notre API suivant le chemin d'accès "model/", ce dernier est spécifier aux niveau de la variable urlpatterns dans le fichier du projet Django "urls.py", et qui est destiné pour renvoyer la réponse de la classe call_model qui implémente la méthode POST.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('qstsapi/', views.qstsapi, name='qstsapi'),
    # Adding a new URL
    path('model/', views.call_model.as_view()),
    path('chat/', views.chatbot),
    path('myAPI/', views.chatbot_api_view, name='chatbot_api')
]
```

Cette dernière sert de la variable model déclaré dans la classe de configuraiton "QstsapiConfig" déclaré dans le fichier "apps.py" de l'application.

5 - Implémentation :

5.B QSTModel :

Le modèle sur Django:

```
from django.apps import AppConfig
from .qmdl.QModel import QModel

class QstsapiConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'qstsapi'
    model = QModel('./qstsapi/qmdl/formData.csv')
```

Ici, notre fichier QModel qui contient le code de la classe QModel est dans le dossier "qmdl" dans la même répertoire de l'application (celui contenant "apps.py")..

Dans le fichier "views.py" nous déclarons la classe call_model qui est appelé lorsque l'utilisateurs soumis le formulaire, celle-ci implémente la méthode post qui prend en paramètre l'objet de la requête, on récupère de ce dernier, si la méthode d'envoi est POST, les réponses aux questions, on ajoute à la réponse, qui est mise sous forme d'une liste, les champs manquant, qui sont optionnels (nom, prénom..), nous déclarons aussi un dictionnaire

5 - Implémentation :

5.B QSTModel :

Le modèle sur Django:

contenant des textes relatifs aux différentes filières..
après l'utilisation du modèle pour la prédiction,nous ajoutons aussi les pourcentage des filières non présentes dans la réponse, qui auront automatiquement un pourcentage de 0, nous cherchons finalement la filière dont l'appartenance de notre étudiant est la plus élevé, nous retournant finalement un template HTML rempli en se basant sur cette filière.et les pourcentage d'appartenance à chaque filière.

5 - Implémentation :

5.B QSTModel :

Le modèle sur Django:

```

def post(self,request):
    if request.method == 'POST':
        optionalFields = ['FName','LName','Email','Major']
        optionalFields.reverse()
        majors = ['IID','GI','GE','IRIC','GPEE']
        #texts to describe each major :
        texts={
            'IID':"Are you hungry for Data ?",
            'GI':"Time to build the next Facebook!",
            'GE':"Autodriving cars said hello!",
            "IRIC":"Watch out, the internet is a dangerous place!",
            "GPEE":"Process, process, and more process!"
        }
        params = request.POST
        #Preprocessing
        x = list(params.values())
        x.insert(0,datetime.now().date())
        if(len(x)<25):
            for optF in optionalFields:
                if optF not in params.keys():
                    x.insert(1,'')
        #predetion
        rsp = QstsapiConfig.model.Evaluate(x)
        #Empty majors:
        for major in majors :
            if major not in rsp.keys():
                rsp[major] = 0.0
        #detect max major:
        max=0
        maxmajor= 'IID'
        for maj in majors:
            if rsp[maj]>max:
                max = rsp[maj]
                maxmajor = maj
        rsp['maxmajor']=maxmajor
        rsp['text']=texts[maxmajor]
        return render(request, 'results.html', rsp)
    
```

5 - Implémentation :

5.B QSTModel :

Le modèle sur Django:

Une partie du fichier "results.HTML", le template à remplir.

```
<section id="results">
    <div class="container pt100 pb90">
        <div class="row">
            <div class="col-12-lg">
                <div class="col-md-6 pt40 pb30">
                    <div class="progress-bars standard transparent-bars" data-animate-on-scroll="on">
                        <h5 class="bold text-center">IID</h5>
                        <div class="progress" data-percent="{{IID}}%">...
                        </div>

                        <h5 class="bold text-center">GI</h5>
                        <div class="progress" data-percent="{{GI}}%">...
                        </div>

                        <h5 class="bold text-center">GE</h5>
                        <div class="progress" data-percent="{{GE}}%">...
                        </div>

                        <h5 class="bold text-center">GPEE</h5>
                        <div class="progress" data-percent="{{GPEE}}%">...
                        </div>

                        <h5 class="bold text-center">IRIC</h5>
                        <div class="progress" data-percent="{{IRIC}}%">...
                        </div>
                    </div>
                </div>
                <div class="col-md-6 ">
                    <h2>Probably, the best major for you is :<br><span class="color">{{maxmajor}}</span></h2>
                    <h5>{{text}}</h5>
                    <br>
                    
                </div>
            </div>
        </div>
    </div>
</section>
```

5 - Implémentation :

5.C Algorithme :

Pour implémenter l'algorithme décrit au paravent, nous avons utilisé le langage de programmation python.

Notre algorithme besoin les coefficients des modules pour les deux années préparatoires c'est-à-dire les notes des 24 modules des 4 semestres , pour cela nous avons définit un dictionnaires qui contient 4 clés qui sont les filières (IID, GI, GPEE, GE, IRIC) et les valeurs sont aussi des dictionnaires qui contient les modules comme des clés et le coefficient de chaque module comme valeur .

```
# Définir un dictionnaire de coefficients pour chaque filière
coefficients = {
    "IID": {"Analyse 1": 7, "Algebre 1": 7, "Mecanique 1": 1, "Physique 1":1,"LC1":2,"informatique 1":7,"Physique 2":1,
             "Algebre 2": 7, "Analyse 2": 7,"Chimie":1,"Informatique 2":7,"LC2":2,"Algebre 3":7,"Analyse3":7,"Mécanique 2":1,
             "Electronique1":1,"Informatique3":7,"LC3":2,"Analyse 4":7,"Math Appliquées":7,"Physique 4":1,"Physique 3":1,
             "Electronique 2":3,"LM":2},
    "GI": {"Analyse 1": 5, "Algebre 1": 5, "Mecanique 1": 1, "Physique 1":1,"LC1":2,"informatique 1":7,"Physique 2":1,
            "Algebre 2": 5, "Analyse 2": 5,"Chimie":1,"Informatique 2":7,"LC2":2,"Algebre 3":5,"Analyse3":5,"Mécanique 2"
            "Electronique1":1,"Informatique3":7,"LC3":2,"Analyse 4":5,"Math Appliquées":5,"Physique 4":1,"Physique 3":1,
            "Electronique 2":1,"LM":2},
    "GE": {"Analyse 1": 1, "Algebre 1": 1, "Mecanique 1": 2, "Physique 1":4,"LC1":2,"informatique 1":2,"Physique 2":4,
            "Algebre 2": 1, "Analyse 2": 1,"Chimie":1,"Informatique 2":3,"LC2":2,"Algebre 3":1,"Analyse3":1,"Mécanique 2":2,
            "Electronique1":4,"Informatique3":3,"LC3":2,"Analyse 4":1,"Math Appliquées":2,"Physique 4":3,"Physique 3":2,
            "Electronique 2":4,"LM":2},
    "GPEE": {"Analyse 1": 2, "Algebre 1": 1, "Mecanique 1": 3, "Physique 1":2,"LC1":1,"informatique 1":1,"Physique 2":1,
              "Algebre 2": 3, "Analyse 2": 3,"Chimie":3,"Informatique 2":3,"LC2":1,"Algebre 3":1,"Analyse3":2,"Mécanique 2":3,
              "Electronique1":2,"Informatique3":2,"LC3":1,"Analyse 4":2,"Math Appliquées":3,"Physique 4":1,"Physique 3":3,
              "Electronique 2":2,"LM":1},
    "IRIC": {"Analyse 1": 2, "Algebre 1": 2, "Mecanique 1": 1, "Physique 1":1,"LC1":2,"informatique 1":3,"Physique 2":2,
              "Algebre 2": 2, "Analyse 2": 2,"Chimie":1,"Informatique 2":3,"LC2":1,"Algebre 3":2,"Analyse3":2,"Mécanique 2":1,
              "Electronique1":2,"Informatique3":3,"LC3":1,"Analyse 4":2,"Math Appliquées":3,"Physique 4":4,"Physique 3":1,
              "Electronique 2":1,"LM":1}
}
```

Activer Windows
Accédez aux paramètres pour activer

5 - Implémentation :

5.C Algorithme :

Après la définition du dictionnaire, On demande à l'utilisateur d'entrer les notes de chaque modules, on utilisant une boucle for pour parcourir notre dictionnaire et puisque les modules restes les même pour chaque filières alors l'utilisateur à demander seulement chaque modules une seule fois c'est pour cette raison on a choisi de parcourir notre conteur par rapport à la filière IID :
les notes entrent par l'utilisateur sont stocké sous format d'un dictionnaire : note

```
# Demander à l'utilisateur d'entrer ses notes pour chaque module
notes = {}
print("Veuillez entrer vos notes pour chaque module :")
for module in coefficients["IID"]:
    note = float(input(f"{module}: "))
    for filiere in coefficients:
        if filiere not in notes:
            notes[filiere] = {}
        notes[filiere][module] = note
```

4 - Implémentation :

4.C Algorithme :

Lorsque le dictionnaire "note" remplit par l'utilisateur , on va maintenant Calculer la moyenne de chaque filière tel que on va multiplié dans un premier temps chaque note par son coefficient pour chaque filière et après on va faire la somme de notre résultat et on va la divisé sur la somme des coefficients et à la fin on va affiché la moyenne pour chaque filière

```
# Calculer La moyenne pondérée pour chaque filière
moyennes_ponderees = {}
for filiere, modules in coefficients.items():
    somme_notes = 0
    somme_coefficients = 0
    for module, coefficient in modules.items():
        note = notes[filiere][module]
        somme_notes += note * coefficient
        somme_coefficients += coefficient
    moyenne_ponderee = somme_notes / somme_coefficients
    moyennes_ponderees[filiere] = moyenne_ponderee
print(f"Moyenne pour la filière {filiere}: {moyenne_ponderee:.2f}")
```

la moyenne de chaque filière

5 - Implémentation :

5.C API :

urls.py

```
from django.urls import path, views
path('gradesapi/', views.gradesapi, name='gradesapi'),
path('algo/',views.calcul_moyenne.as_view()),
```

Pour notre model qui calcul la moyenne général pour chaque filière ,
on a définit deux chain pourdeux différentes views . Ce qui laisse à chaque l'appel de la vue 'views.gradesapi' sera appelée pour traiter la requête quand on accéde au url"gradesapi/". La même chose pour l'url "algo/" qui est associé à la vue 'views.calcul_moyenne '

Remarque : La méthode **.as_view()** est utilisée pour convertir la vue basée sur une classe en une vue utilisable

views.py

Dans le fichier "views.py" nous déclarons la classe `calcul_moyenne` qui est appelé lorsque l'utilisateurs soumis le formulaire des notes , celle-ci implémente la méthode post qui prend en paramètre l'objet de la requête, on récupère de ce dernier, si la méthode d'envoi est POST,

5 - Implémentation :

5.C API :

Après l'utilisation du modèle pour calculer la moyenne générale pour chaque filière ,nous ajoutons aussi que ces moyennes là seront affiché par ordre croissant , nous retournant finalement un template HTML qui affiche le résultat sous formes d'un tableau des filières avec leurs moyennes pour chaque étudiant .

```

class calcul_moyenne(APIView):
    def post(self,request):
        if request.method == 'POST':
            notes = {}
            if len(list(request.POST.values()))==24:
                for key in request.POST.keys():
                    notes[key]=float(request.POST[key])

            # Création de l'instance du modèle
            calcul_moyenne = CalculMoyenneFiliere()
            # Appel des méthodes pour calculer les moyennes
            moyennes = calcul_moyenne.calculer_moyennes(notes)
            maxMoyMaj = ''
            maxMoy = 0
            for maj in moyennes.keys():
                if moyennes[maj]>maxMoy:
                    maxMoy = moyennes[maj]
                    maxMoyMaj = maj
            moyennes['maxMaj']= maxMoyMaj
            texts={

                'IID':'Are you hungry for Data ?',
                'GI':'Time to build the next Facebook!',
                'GE':'Autodriving cars said hello!',
                'IRIC':'Watch out, the internet is a dangerous place!',
                'GPEE':'Process, process, and more process!'
            }
            majCuts = {
                'IID':'Informatique et Ingénierie des Données (IID)',
                'GI':'Genie Informatique (GI)',
                'GE':'Genie Electrique (GE)',
                'IRIC':'Ingenierie des Réseaux Intelligent et Cybersecurité (IRIC)',
                'GPEE':'Genie des Procédés et des Energies Renouvelables (GPEE)',
            }
            moyennes['maxMajName']=majCuts[maxMoyMaj]
            moyennes['text']=texts[maxMoyMaj]

        return render(request, 'gradesFormResults.html', moyennes)

def gradesapi(request):
    # index c'est le nom de la page html dans le dossier views
    template = loader.get_template('gradesForm.html')
    return HttpResponseRedirect(template.render({}, request))

```

5 - Implémentation :

5.C API :

GradesForm.html

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link href="https://fonts.googleapis.com/css?family=Roboto:400,700,900&display=swap" rel="stylesheet">

    <link rel="stylesheet" href="{% static 'fonts/icomoon/style.css' %}">
    {% include 'head.html' %}

    <!-- Style -->
    <link rel="stylesheet" href="{% static 'css/style.css' %}">

    <title> OrientAi : Grades Form </title>
  </head>
  <body data-fade-in="true">

    <div class="pre-loader">
      <div></div>
    </div>
    {% include 'header.html' %}

    <script>
      window.addEventListener('load', function() {
        var targetSection = document.getElementById('here');
```

Pour utiliser notre modèle, nous créons une simple page en HTML qui sert à grouper les réponses de l'utilisateur et les envoyer avec la méthode POST à notre API suivant le chemin d'accès "algo/"

5 - Implémentation :

5.C API :

Gradeformresults.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>OrientAi: Results </title>
    <link rel="stylesheet" href="{% static 'fonts/icomoon/style.css' %}">
    {% include 'head.html' %}

    <!-- Style -->
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>

<body data-fade-in="true">

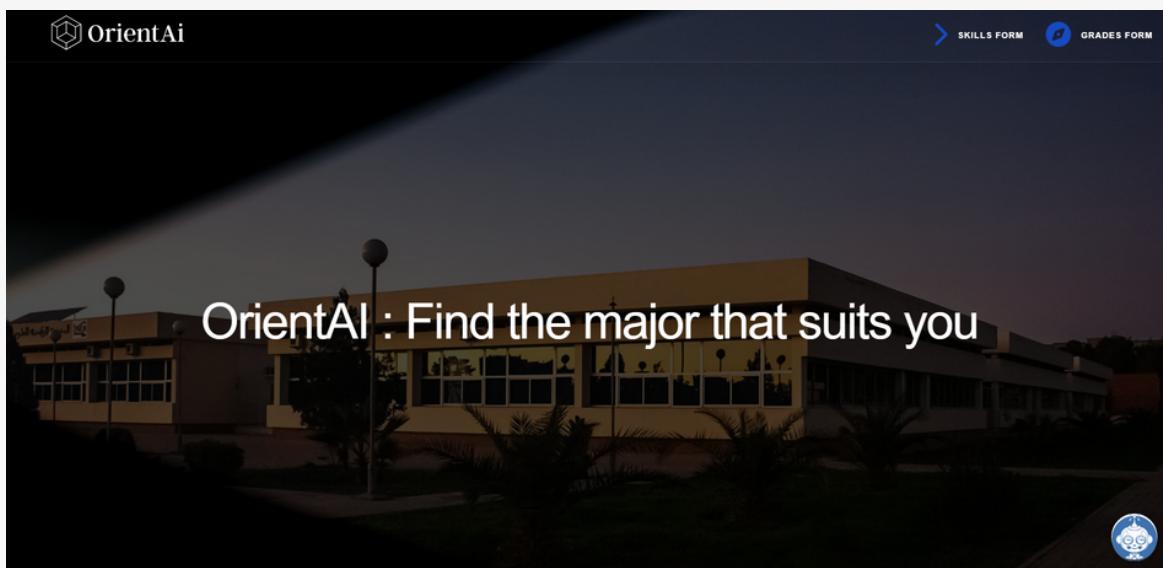
    <div class="pre-loader">
        <div></div>
    </div>
    <script>
        window.addEventListener('load', function() {
            var targetSection = document.getElementById('resultss');
            if (targetSection) {
                targetSection.scrollIntoView();
            }
        });
    </script>

    {% include 'header.html' %}

    <!-- Hero -->
    <section id="hero" class="hero-fullscreen parallax" data-overlay-dark="7">
```

6. Expérience Utilisateur:

L'accueil :



OrientAI Project
Students data to help students future!

IMA ChatBot

IMA: is a computer program that uses artificial intelligence and NLP (Natural Language Processing).

Its goal is to assist students in discovering their educational orientation choices.

The IMA ChatBot is designed to interact with students online, asking them questions about their interests, skills, and goals, and providing them with information about careers or study programs that may suit them.

It also allows students to ask questions about the National School of Applied Sciences in Khouribga.

Skills Form

This form suggests the suitable field of study for the student based on their skills, by answering a set of questions related to the skills associated with the IID field.

Grades Form

This form once again suggests the suitable major to the student, but this time based on their grades. The student needs to enter the grade for each module of the 4 semesters, and the program will rank the majors according to their calculated averages.

Our Team

"If I have seen further, it is by standing on the shoulders of giants." — Isaac Newton



Guerrab Mouna

| 2nd Year Data and Software Engineering Student @ ENSA Khouribga



El Amraoui Ilyas

| 2nd Data and Software Engineering Student @ ENSA Khouribga.



El Jaouhari Mohamed

| 2nd Data and Software Engineering Student @ ENSA Khouribga.



Matouk Afaf

| 2nd Data and Software Engineering Student @ ENSA Khouribga.



To build this
We used the following
skills



Visit
Our school : ENSA KHOURIBGA



Formulaire des compétences :



Personal Information

Filling these fields is fully optional

First Name :

Last Name :

Email :

Major (if you are already post preparatory classes only) :

If you are filling this form to populate the data, please answer with respect to skills and knowledge you had when you were at API2.

1. What is your interest in technology?

- a. I love it
- b. I am not sure
- c. I don't like it

2. Do you enjoy programming?

- a. Yes
- b. No

OrientAi

SKILLS FORM GRADES FORM

b. No

c. I am not sure

20. What is your experience with project management?

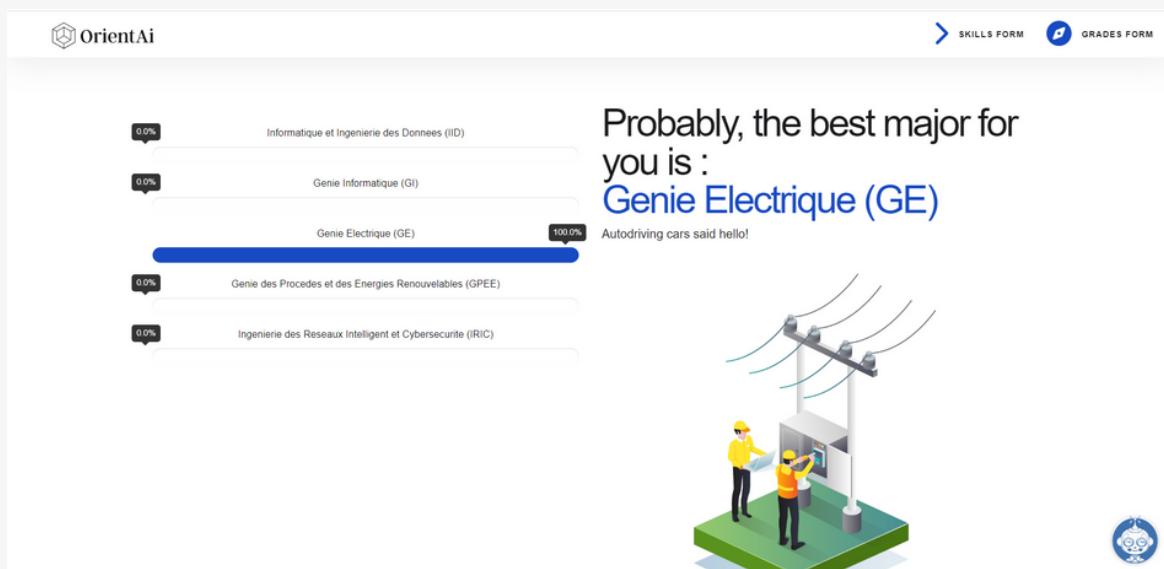
a. I am very familiar with it

b. I know the basics

c. I have little to no experience

FIND OUT

Le reste est similaire à celle de la page Accueil.
En remplissant le formulaire, nous serons envoyer vers la page suivante où l'étudiant aura un pourcentage pour chaque filière.



Formulaire des notes :

Cette page permet l'étudiant à entrer ses notes pour avoir sa propre moyenne pour chaque filière comme l'exemple suivant le montre :

Enter your grades

Analyse 1	12	Algèbre 1	13
Mécanique 1	11	Physique 1	11
LC 1	15	Informatique 1	14
Physique 2	10	Algèbre 2	17
Analyse 2	19	Chimie	18
Informatique 2	16	LC 2	13
Algèbre 3	17	Analyse 3	15
Mécanique 2	18	Électronique 1	11
Informatique 3	11	LC 3	10
Analyse 4	16	Math Appliquées	18
Physique 4	15	Physique 3	18
Électronique 2	15	LM	19

SEND

Results :

Informatic et Ingénierie des Données (IID)	15.15625
Genie Informatique (GI)	15.012820512820513
Genie Electrique (GE)	14.607843137254902
Ingénierie des Réseaux Intelligent et Cybersecurité (IRIC)	15.022222222222222
Genie des Procedes et des Energies Renouvelables (GPEE)	15.340425531914894

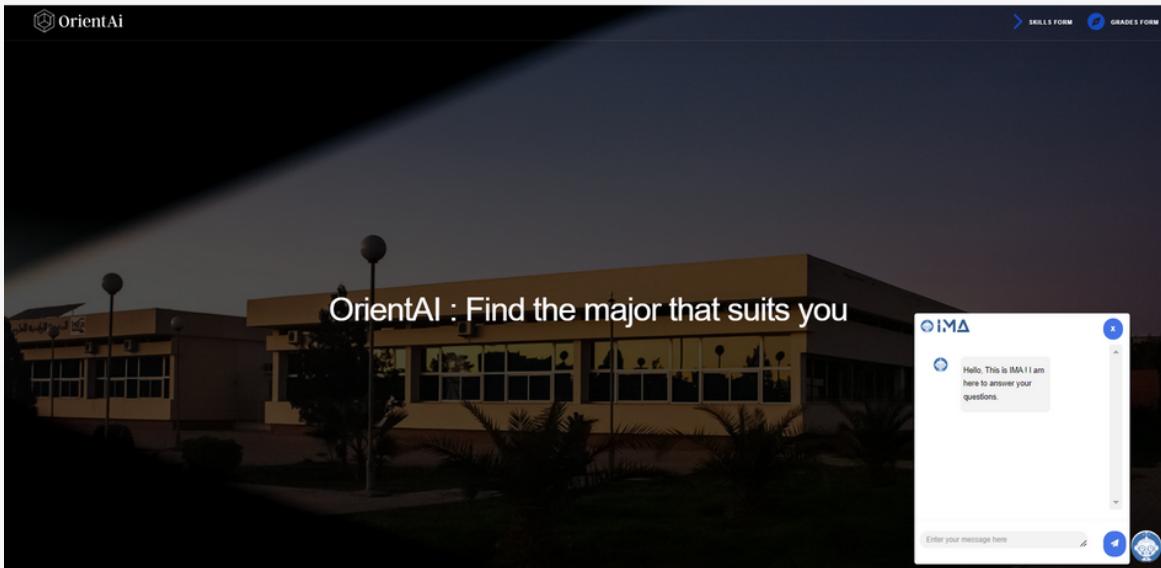
Probably, the best major for you is :
Genie des Procedes et des Energies Renouvelables (GPEE)

Process, process, and more process!

Process, process, and more process!

Chatbot :

Notre chatbot est intégrée dans toute les pages, en cliquant sur l'icone, une discussion est ouverte comme suivant :



Pour cette version, un chatbot fonctionnelle est non disponible dans le projet puisque L'entraînement de ce chatbot présente des difficultés au niveau des processeurs locaux en raison de la température élevée atteinte par le CPU (80°C) et du temps d'exécution long (près de 22 heures pour un CPU).

L'entraînement sur un GPU de type T4 sur Google Colab nous a permis d'obtenir notre modèle, mais celui-ci ne fonctionne pas correctement lorsqu'on essaie de le faire fonctionner sur un CPU local. Cependant, l'utilisation d'un CPU est nécessaire car l'infrastructure de déploiement est composée de CPUs et non de GPUs.

Nous sommes entrain de régler le problème, et nous vous promettons que cela sera réglé avant la soutenance planifiée.

Nous montrerons des Questions-Réponses de notre modèle utilisée sur le GPU mentionné à fin de confirmer notre travail et motivation pour compléter le projet dans la meilleure forme possible.

Merci pour votre compréhension,
Cordialement.

Chatbot : (Execution sur GPU T4 - Google Colab)

```
✓ [20] 1 def ask(sentence):  
2     result, sentence = evaluate(sentence,encoder,decoder)  
3  
4     print('Question: %s' % (sentence))  
5     print('Predicted answer: {}'.format(result))
```

```
✓ [21] 1 ask("hello")
```

Question: hello
Predicted answer: hello, how can i help ?

```
[22] 1 ask("how are ya ?")
```

Question: how are ya
Predicted answer: happy to be here for assistance. how can i help you ?

```
[23] 1 ask("what is your name ?")
```

Question: what is your name
Predicted answer: my name is ima

```
✓ [24] 1 ask("who are your creators ?")
```

↳ Question: who are your creator
Predicted answer: iid student : afaf matouk, mouna guerrab, ilyas el amrani, mohamed el jaouhari

7. Vision :

- Améliorer le taux de satisfaction des étudiants en termes de choix de filière pour être plus productifs et créatifs dans leur travail.
- Aider les étudiants de faire le bon choix en termes de formation pour diminuer le taux d'erreurs
- Améliorer la satisfaction des étudiants et leurs familles au niveau du système scolaire au Maroc.
- Améliorer notre chabot IMA et le rendre utile au réseau ENSA dans un premier lieu.



8. Conclusion :

Ce projet nous a permis d'acquérir des connaissances au niveau de technologie web (Django), Apprentissage profond (Chatbot) et Apprentissage automatique (Model d'orientation).

Comme clôture pour notre projet, nous sommes fiers d'avoir une chance d'améliorer le problème de l'orientation grâce à cette solution que nous avons développée avec motivation et dévouement pendant 4 mois pour notre établissement et tous les départements impliqués.

