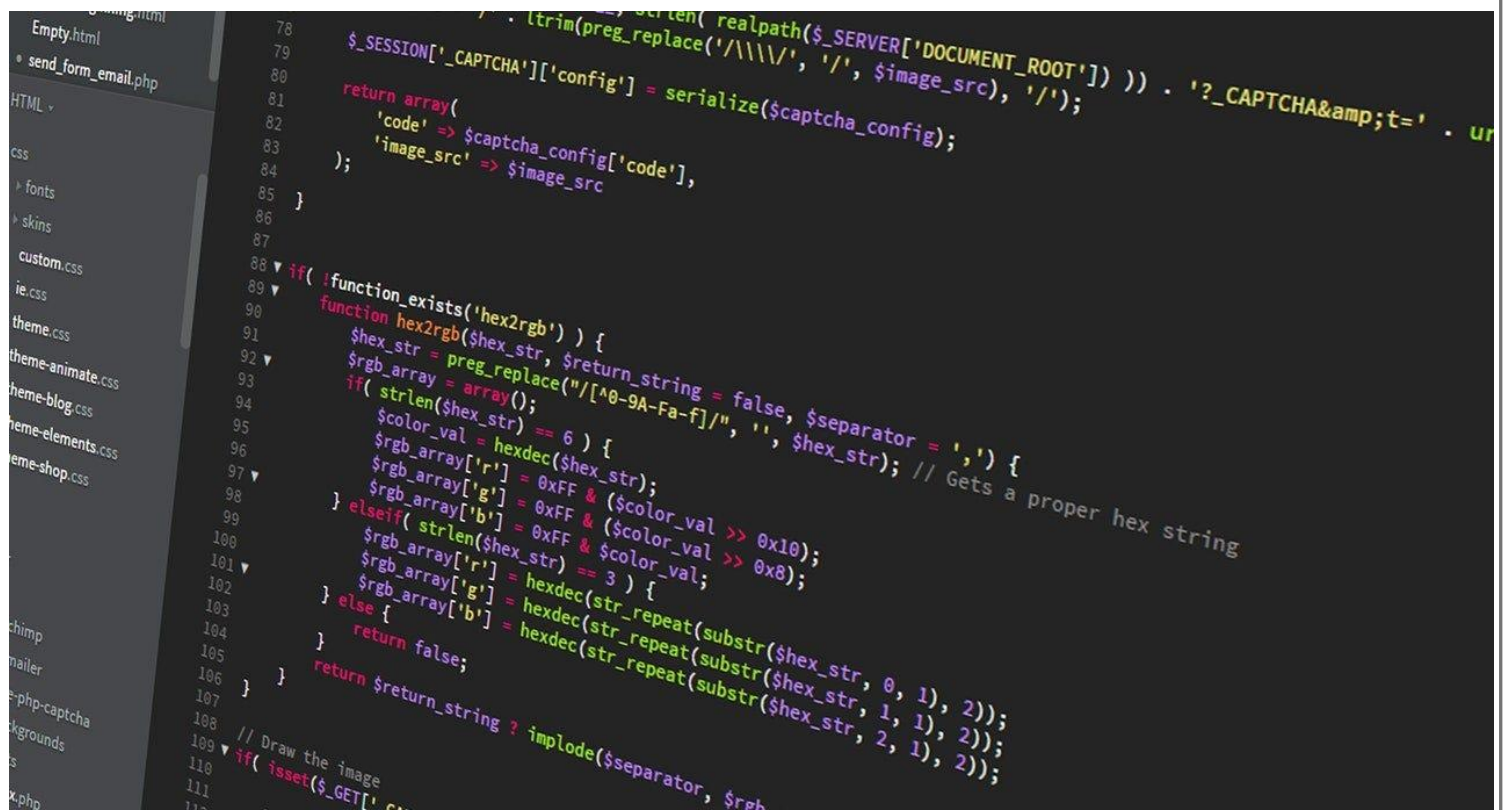


Documentation Fonctionnelle

→ ConversaSD



```
78 // Trim the path and replace backslashes with forward slashes
79 $realpath = trim(preg_replace('/\\\\\\/', '/', $image_src), '/');
80
81 $SESSION['_CAPTCHA']['config'] = serialize($captcha_config);
82
83 return array(
84     'code' => $captcha_config['code'],
85     'image_src' => $image_src
86 );
87
88 if ( !function_exists('hex2rgb') ) {
89     function hex2rgb($hex_str, $return_string = false, $separator = ',') {
90         $hex_str = preg_replace("/[^0-9A-Fa-f]/", '', $hex_str); // Gets a proper hex string
91         $rgb_array = array();
92         if ( strlen($hex_str) == 6 ) {
93             $color_val = hexdec($hex_str);
94             $rgb_array['r'] = 0xFF & ($color_val >> 0x10);
95             $rgb_array['g'] = 0xFF & ($color_val >> 0x08);
96             $rgb_array['b'] = 0xFF & $color_val;
97         } elseif ( strlen($hex_str) == 3 ) {
98             $rgb_array['r'] = hexdec(str_repeat(substr($hex_str, 0, 1), 2));
99             $rgb_array['g'] = hexdec(str_repeat(substr($hex_str, 1, 1), 2));
100             $rgb_array['b'] = hexdec(str_repeat(substr($hex_str, 2, 1), 2));
101         } else {
102             return false;
103         }
104         return $return_string ? implode($separator, $rgb_array) : $rgb_array;
105     }
106 }
107
108 // Draw the image
109 if ( !isset($_GET['code']) ) {
110     // ...
111 }
```

Created By :

- Gabriel BREGAND
- Zouhour ABASSY
- Cordelia GUY
- Najoua AJOUIRJA



Table des matières

Documentation Fonctionnelle	1
A. Prérequis	3
1. Mise en place de kaggle	3
2. Mise en place de HuggingFace	3
3. Ajouter votre base de données ZIP à l'espace de travail.....	3
4. Configurer les settings du notebook	3
5. Lancer l'intégralité du notebook (« Run all »)	3
B. Paramétrages	4
Paramètres développeur	4
Paramètres utilisateur	5
C. Nettoyage des fichiers : ConversaSD_clearfil	7
Décompression des fichiers	7
Nettoyage des noms de fichiers.....	7
Conversion et traitement des documents	7
D. RAG (Génération augmentée par récupération.)	8
Création des Chunks.....	8
Création de l'indexation	8
E. Chatbot.....	9
Construction du prompt et génération de réponse	9
Exemple de prompt construit :.....	9
Authentification Hugging Face.....	10
Chargement des ressources.....	10
F. Configuration des logs (ConversaSD_log.py)	11
Création du logger personnalisé	11
G. Main.....	12
Fonction principale du script.....	12



A. Prérequis

1. Mise en place de kaggle

Assurez-vous d'avoir un compte [Kaggle](#) pour pouvoir exécuter le code sur la plateforme. Si ce n'est pas déjà le cas, voir la documentaire utilisateur.

2. Mise en place de HuggingFace

Un compte [HuggingFace](#) sera également nécessaire pour avoir accès aux modèles all-MiniLM-L6-v2 (embedding) et mistralai/Mistral-7B-Instruct-v0.3 (génératif) grâce un token que vous aurez créé au préalable.

3. Ajouter votre base de données ZIP à l'espace de travail

1. Dans le volet de droite en dessous de 'Input', cliquez sur Cliquer sur « +add Input » -> « New Dataset »
2. Choisissez l'onglet **Your Datasets**.
3. Repérez votre dataset « zip-supports-programmation » (ou créez-le au préalable et récupérer).
4. Cliquez sur **Add** à côté de ce dataset : il sera monté sous « /kaggle/input/supports-programmation. »
5. Vous pouvez vérifier en ajoutant une cellule tout en haut :

```
import os
print(os.listdir('/kaggle/input/supports-programmation'))
```

4. Configurer les settings du notebook

Si vous avez bien fait la vérification par numéro de téléphone de votre compte

1. En haut à droite, cliquez sur **Settings**
 - **Accelerator** → **GPU**
 - dans la liste déroulante, choisissez **T4 × 2**

5. Lancer l'intégralité du notebook (« Run all »)

Avant de lancer le notebook il est indispensable d'avoir un [Token Huggingface](#)

1. Dans la barre de menus du notebook, cliquez sur **Run** → **Run all cells**.
2. Le notebook se lancera. Cependant veillez à avoir votre clé token de HuggingFace.



B. Paramétrages

Paramètres développeur

ConversaSD_parametre.py

Emplacements des fichiers

Nom	Description	Exemple
emplacement_code	Emplacement du main	"C:/ConveraSD/"
zip_file	Chemin du fichier ZIP contenant les fichiers à traiter. (par défaut au même emplacement que le code + fichier)	emplacement_code + "data_zip/Supports programmation.zip" ("C:/ConveraSD/data_zip/Supports programmation.zip")
extract_folder	Dossier où seront extraits les fichiers ZIP. (par défaut au même emplacement que le code + direction du fichier)	emplacement_code + "extract_file/" ("C:/ConveraSD/extract_file/")
folders	Dossiers ciblés dans l'archive ZIP. Utilisés pour filtrer les fichiers pertinents.	["Supports programmation/S1", "Supports programmation/S2"]
output_txt_file	Répertoire de sortie des fichiers transformés en texte brut (fichier .txt).	emplacement_code + "data/extracted_data/" ("C:/ConveraSD/data/extracted_data/")

Fichiers d'indexation FAISS

Nom	Description	Exemple
index_path	Chemin du fichier FAISS sauvegardant l'index vectoriel.	output_txt_file + "faiss.index" ("C:/ConveraSD/data/extracted_data/faiss.index")
mapping_path	Fichier pickle pour associer chaque chunk à son contenu.	output_txt_file + "chunks.pkl" ("C:/ConveraSD/data/extracted_data/chunks.pkl")

OCR

Nom	Description	Exemple
file_pyesseract	Commande ou chemin vers l'exécutable Tesseract OCR.	"tesseract" ou "C:/Program Files/Tesseract-OCR/tesseract.exe"



lang_orc	Langues utilisées par l'OCR (code ISO, séparés par +).	"fra+eng" (français et anglais)
----------	--	---------------------------------

Modèles

Nom	Description	Exemple
embed_model_name	Nom du modèle d'embedding (vecteur sémantique) Hugging Face.	'all-MiniLM-L6-v2'
gen_model	Modèle génératif utilisé pour répondre aux questions.	"mistralai/Mistral-7B-Instruct-v0.3"

Paramètres utilisateur par défaut

Nom	Description	Exemple	Argument CLI
chunk_size_default	Nombre de tokens par chunk (bloc de texte).	256	--chunk_size
chunk_overlap_default	Nombre de tokens de chevauchement entre deux chunk.	64	--chunk_overlap
max_tokens_default	Nombre maximum de tokens retournés par le chatbot.	128	--max_tokens
topk_default	Nombre de chunks les plus pertinents à prendre en compte.	1	--topk

Paramètres utilisateur

ConversaSD_main.py

Paramètres utilisateur — get_config()

La fonction `get_config()` permet de configurer dynamiquement le comportement du programme via des **arguments en ligne de commande**. Elle renvoie un dictionnaire contenant les valeurs choisies, qui seront utilisées par les modules du pipeline.

Exemple d'utilisation

Bash :

```
python ConversaSD_main.py --log-level info --unzipfil True --chunk_size 128
```

Liste des paramètres

Abrégé	Nom long	Type	Valeur par défaut	Description
-ll	--log-level	str	"debug"	Niveau de journalisation à adopter. Choix possibles : debug,



				info, warning, error, critical.
-lf	--log-file	str	"log_execution.log"	Nom du fichier dans lequel les logs seront enregistrés.
-uz	--unzipfil	bool	False	Si True, décompresse automatiquement les fichiers ZIP dans le dossier prévu.
-et	--extractetxt	bool	True	Si True, extrait et convertit les fichiers PDF, CSV, etc. en texte brut .txt.
-cs	--chunk_size	int	parametre.chunk_size_default (256)	Nombre de tokens par segment de texte (chunk).
-co	--chunk_overlap	int	parametre.chunk_overlap_default (64)	Chevauchement en tokens entre deux segments consécutifs.
-tk	--topk	int	parametre.topk_default (1)	Nombre de segments (chunks) les plus pertinents à utiliser pour générer la réponse.
-mt	--max_tokens	int	parametre.max_tokens_default (128)	Nombre maximum de tokens autorisés dans une réponse générée par le chatbot.



C. Nettoyage des fichiers : ConversaSD_clearfil

ConversaSD_clearfil.py

Automatisation de l'extraction et de la transformation de documents pédagogiques en fichiers .txt nettoyés, exploitables par le chatbot *ConversaSD*.

Décompression des fichiers

Fonction unzip

Cette fonction permet de décompresser automatiquement une archive .zip contenant des ressources (PDF, notebooks, CSV) dans un dossier de travail temporaire. Elle vérifie que le fichier ZIP existe, crée un dossier si besoin, puis extrait les fichiers. Toutes les étapes sont suivies via un système de logs, ce qui facilite le diagnostic en cas d'erreur. Cette fonction est la porte d'entrée du traitement documentaire automatisé.

Nettoyage des noms de fichiers

Fonction clean_filename

Cette fonction standardise les noms de fichiers extraits pour qu'ils soient compatibles avec tous les systèmes d'exploitation. Elle supprime les accents, les caractères spéciaux ou invisibles, et remplace tout symbole non standard par un underscore (_). Cela permet d'éviter des erreurs lors de la sauvegarde ou du traitement des fichiers. C'est une étape préparatoire cruciale avant l'enregistrement final.

Conversion et traitement des documents

Fonction convert_fil_to_individual_texts

Cette fonction est le cœur du traitement automatique. Elle parcourt tous les fichiers d'un ou plusieurs dossiers, détecte leur format (PDF, .ipynb, .csv), puis extrait uniquement les informations utiles.

- Pour les PDF, le texte est extrait avec fitz. Si des images sont présentes, un OCR avec Tesseract est appliqué pour récupérer le texte contenu dans ces images.
- Pour les notebooks Jupyter, seules les cellules de type code et markdown sont conservées. Cela permet d'isoler les blocs pédagogiques intéressants pour le chatbot.
- Pour les CSV, les lignes sont converties en phrases textuelles simples, ligne par ligne, en conservant la structure.
- Tous les autres formats sont ignorés, mais le système reste extensible : il est prévu d'ajouter de nouveaux types si besoin.

Chaque fichier est ensuite nettoyé (encodage, ponctuation, caractères spéciaux) pour produire un texte exploitable propre et cohérent. Le nom du fichier .txt est généré à partir de l'origine du fichier, ce qui permet de suivre facilement la provenance. Tous les traitements sont enregistrés dans les logs, avec un système de détection des doublons pour éviter de retraiter deux fois le même document.



D. RAG (Génération augmentée par récupération.)

ConversaSD_rag.py

Création du [RAG](#) pour optimiser le résultat du modèle avec l'ajout de contexte ici via information contenue dans des documents.

Création des Chunks

Fonction chunk_text

Prépare le texte en le segmentant en chunks, soit découpage pour passer d'un long texte à ensembles de texte en segments plus petits. Les chunk sont plus adaptés à l'analyse sémantique et à l'indexation vectorielle.

La segmente de texte est paramétrable, on peut choisir le nombre de [tokens](#) pour chaque segmentation faite (**chunk_size**). Nous avons choisi 256 tokens pour ne pas faire de trop grand chunk.

Nous pouvons aussi choisir quel est le nombre maximum de tokens que le chunk peut déborder sur celui d'avant ou après (**chunk_overlap**) pour permettre de mieux saisir le contexte général du texte, le choix de ce paramètre doit s'appuyer sur la taille du chunk.

Pour séparer de manière optimale le texte, il est important de choisir un [pipeline](#) adapter à la langue nous avons choisi [fr_core_news_md](#) sachant que le document semble être tous en français il est possible cependant que les partie code soit moins bien interpréter

Création de l'indexation

Fonction build_index

Cette fonction gère la création de l'indexation des documents texte. Elle regroupe plusieurs étapes clés allant du découpage des documents (**chunk**), transforme ces chunks en [vecteurs sémantiques](#) via un modèle d'embedding, pour générer l'indexation.

Il est important de paramétrer le modèle d'embedding (`embed_model_name`), nous avons, sachant que ce programme est créé dans un contexte universitaire, un modèle gratuit. Nous aurions pu choisir [Llama](#) mais nous avons choisi [all-MiniLM-L6-v2](#) pour l'index, il semble plus rapide et léger pour l'embedding de phrases courtes.



E. Chatbot

ConversaSD_chatbot.py

Ce script gère le dialogue avec l'utilisateur en s'appuyant sur les éléments préalablement préparés dans le module *ConversaSD_rag.py*.

Ce module fait appel à un modèle génératif (LLM) capable de produire une réponse en langage naturel, enrichie par des informations extraites du corpus indexé.

Concrètement, le script :

- Authentifie l'utilisateur Hugging Face (si besoin),
- Charge les ressources nécessaires : index vectoriel, modèles et tokenizer,
- Transforme la question posée en vecteur sémantique,
- Recherche les chunks les plus proches dans l'index FAISS,
- Construis un prompt enrichi avec ces passages,
- Génère une réponse concise en français grâce à un LLM.

Construction du prompt et génération de réponse

Fonction answer

Paramètre :

- Query (La question posée par l'utilisateur.)
- Idx (Index de similarité vectorielle utilisé pour rechercher les passages les plus proches query)
- Chunks (Liste de chunks)
- emb_model (Modèle utilisé pour transformer query en vecteur d'embedding pour la recherche de similarité)
- tokenizer (Convertir le prompt en tenseur, prêt à être lu par le modèle de génération)
- model (Modèle de génération de texte)
- max_new_tokens (Nombre maximum de tokens que le modèle peut générer pour la réponse)
- topk (nombres de chunks pertinents utilisés)

Cette fonction réalise les étapes finales du processus de question/réponse :

- Appelle *retrieve_top(...)* pour récupérer les passages les plus utiles,
- Concatène ces extraits pour construire un prompt clair (limite 512 tokens),
- Crée une requête avec le tokenizer et l'envoie au modèle *AutoModelForCausalLM*,
- Récupère le texte généré, extrait uniquement la partie après "Réponse :", et la retourne avec le contexte utilisé.

Exemple de prompt construit :

Question : "parle moi de str"

Contexte : (extraits de documents...)

Réponse : "str est un type de données en Python qui représente une chaîne de caractères."



Authentication Hugging Face

Fonction ensure_login

Cette fonction vérifie si l'utilisateur est bien connecté à Hugging Face. Si ce n'est pas le cas, elle lance une procédure de login (avec login()), puis attend jusqu'à ce que la connexion soit effective.

L'objectif est de permettre le téléchargement de modèles hébergés sur le hub Hugging Face.

Chargement des ressources

Fonction load_resources

Cette fonction charge tous les éléments nécessaires au bon fonctionnement du chatbot :

- L'index vectoriel
- La liste des chunks
- Le modèle d'embedding pour encoder les questions,
- Le tokenizer associé au LLM utilisé,
- Le modèle de génération

Tous ces éléments sont retournés pour être exploités dans la fonction answer().

Recherche des chunks pertinents

Fonction retrieve_top

Transforme la question de l'utilisateur (query) en vecteur, grâce au même modèle d'embedding que celui utilisé lors de l'indexation.

Elle interroge ensuite l'index FAISS pour retrouver le(s) chunk(s) le(s) plus proche(s) sémantiquement de la question. Ce sont eux qui serviront de contexte pour le LLM.

L'objectif est de contextualiser la question avec des éléments pertinents extraits de la base documentaire.



F. Configuration des logs (ConversaSD_log.py)

ConversaSD_log.py

Cette fonction configure un logger personnalisé pour l'application ConversaSD, en sauvegardant automatiquement les événements dans un fichier texte, tout en les affichant en temps réel dans la console. Voici les étapes clés de la fonction :

1. *Création du dossier de log*
2. *Création du fichier log*
3. *Initialisation du logger*
4. *Formatage des logs*
5. *Double sortie des logs (enregistrement dans un fichier et vue dans la console)*
6. *Retour*

Création du logger personnalisé

Fonction setup_logger

Cette fonction met en place un système de logs structuré pour suivre le comportement de tous les scripts de ConversaSD.

Elle crée automatiquement un dossier logs/ dans le répertoire du projet et y enregistre un fichier .log daté à chaque exécution (ex : 20250624_143215_log_execution.log). Ce fichier contient des informations temporelles, des niveaux de gravité (INFO, DEBUG, ERROR...) et des messages personnalisés décrivant ce que fait le programme à chaque instant.

Le logger fonctionne en double sortie :

- Dans la console pour voir en direct ce qu'il se passe,
- Dans un fichier texte pour garder une trace durable des exécutions.

Ce mécanisme est essentiel pour :

- déboguer efficacement,
- suivre les traitements réalisés (extraction, nettoyage, conversions...),
- tracer les erreurs automatiquement, sans dépendre d'une vérification manuelle.

Grâce à ce module, l'ensemble de la chaîne de traitement documentaire est traçable, transparente et maintenable, ce qui est indispensable dans un projet de traitement automatisé de données.



G. Main

ConversaSD_main.py

→ Script principal qui gère l'ensemble du projet ConversaSD.

Ce fichier est le point d'entrée du programme. Il permet de gérer tout le processus, depuis la préparation des documents jusqu'à l'interaction avec l'utilisateur final via un chatbot RAG (Retrieval-Augmented Generation). Le script est conçu pour être compatible avec un environnement local comme avec Kaggle.

Fonction principale du script

Ce script pilote toutes les étapes suivantes :

1. **Chargement des paramètres utilisateurs** via des arguments en ligne de commande (argparse), comme la taille des chunks, le nombre de chunks à fournir au chatbot (topk), ou le nombre de tokens maximum par réponse.
2. **Décompression automatique des fichiers**, si besoin, depuis un fichier .zip contenant des documents sources (.pdf, .csv, etc.).
3. **Conversion des fichiers** (.pdf, .csv, etc.) en fichiers .txt exploitables.
4. **Création de l'index sémantique** :
 - Les documents .txt sont transformés en chunks,
 - Ces chunks sont ensuite vectorisés avec un modèle d'embedding (ici, all-MiniLM-L6-v2),
 - Le tout est sauvegardé dans un index FAISS + un mapping.
5. **Chargement sécurisé des ressources** nécessaires au chatbot (modèle de génération, tokenizer, index vectoriel...).
6. **Boucle d'interaction utilisateur** : un chatbot RAG qui répond aux questions en exploitant les chunks les plus pertinents retrouvés dans les documents sources.

