École Nationale Supérieure des Arts et Métiers

ENSAM Meknès – Université Moulay Ismaïl

Cycle Ingénieur – Filière IATD-SI
(Ingénierie de l'Intelligence Artificielle et des Technologies de la Donnée pour les Systèmes Industriels)

Rapport de Projet de Fin d'Année

SmartWebScraper-CV

Application intelligente d'annotation de pages web par Computer Vision, OCR, NLP et LLM

Présenté par :
DJERI-ALASSANI OUBENOUPOU
&
EL MAJDI WALID

Encadré par :
Monsieur le Professeur **Tawfik MASROUR**

Date :16 Juin 2025

Table des matières

Page À propos et Remerciements

Ce projet a été réalisé dans le cadre de la formation d'ingénieur à l'ENSAM Meknès, au sein de la filière IATD-SI. Il représente l'aboutissement de plusieurs mois de réflexion, d'exploration technologique, de prototypage, et de tests intensifs.

Je tiens à adresser mes plus sincères remerciements à :

— Monsieur Tawfik Masrour, professeur encadrant, pour sa disponibilité, sa bienveillance, et ses conseils rigoureux tout au long de ce travail.

— L'ensemble de l'équipe pédagogique de l'ENSAM Meknès, pour la qualité de leur enseignement et leur accompagnement durant notre parcours.

— Nos camarades de classe de projet pour leurs échanges constructifs, critiques bienveillantes et idées inspirantes.

— Et bien sûr, à toutes les personnes qui, de près ou de loin, ont contribué au succès de ce projet.

Ce travail marque une étape importante de notre formation, mais aussi une base concrète sur laquelle je compte bâtir des solutions encore plus ambitieuses au croisement de la vision par ordinateur, du traitement du langage et des systèmes intelligents.

Chapitre 1

Introduction

1.1 Contexte général

Dans un monde où le contenu numérique est massivement généré, stocké et diffusé via des interfaces web, la capacité à extraire, comprendre, organiser et exploiter automatiquement ces informations devient cruciale. C'est dans cette perspective que s'inscrit ce projet : SmartWebScraper-CV, une application intelligente combinant Computer Vision, OCR, NLP et LLM pour extraire, comprendre et structurer automatiquement le contenu de pages web capturées sous forme d'images.

Le projet trouve son utilité aussi dans les probleme se scrapping textuelle du à :

L'obfuscation du code HTML.

Obfuscation JavaScript lourde .

Texte rendu sous forme d'image(PDF„ etc.).

Interfaces complexes ou graphiques.

1.2 Objectif du projet

Ce projet a pour objectif de scraper visuellement une page web, c'est-à-dire :

— capturer l'image d'un site,

— détecter automatiquement les zones d'intérêt (titre, pub, footer, etc.),

— extraire le texte des zones conservées,

— et permettre une interaction intelligente avec ce texte (résumé, questions, entités, traduction...).

Le tout s'intègre dans une application complète développée en Flask, avec deux profils :

— utilisateur final, qui peut interagir avec l'image et le texte extrait,

— administrateur, qui supervise, corrige et valide les annotations, prépare les données pour le fine-tuning, et évalue la qualité du système.

1.3 Technologies clés utilisées

— **Recolte d images** : Capture des pages web via Selenium pour générer un dataset d'images, utilisation de SerpAPI pour collecter des URLs depuis Google sans blocage, et undetected-chromedriver pour naviguer de manière indétectable en mode headless (Navigateur invisible, optimisé pour l'automatisation.) .

— **Annotation d images** : Dataset annoté manuellement via Roboflow.

— **Computer Vision** : Detectron2 (Faster R-CNN), annotations COCO, training supervisé sur dataset annoté Roboflow.

— **OCR** : PaddleOCR, traitement avancé de zones, découpe verticale, prétraitement d'image.

— **NLP** : NLTK, TF-IDF, Word2Vec, spaCy, clustering, résumé extractif, moteur de QA, NER.

— **LLM** : Gemini API (Google Cloud), Mistral via Ollama en local.

— **Web App** : Flask, HTML5 canvas pour annotation manuelle, interface responsive.

1.4 Historique du développement

Le projet s'est développé sur plusieurs mois, avec des phases successives :

1. Obtention de $\tilde{3}000$ images via un scraping visuel avec Selenium .
2. Premiers tests d'Annotation des image via HTML avec l architecture DOM (une représentation en arborescence d'une page HTML, créée par le navigateur. par Selenium et JavaScript via la méthode getBoundingClientRect) (inefficace sur les sites modernes protégés).
3. Annotation manuelle sur Roboflow, puis fine-tuning d'un détecteur de Facebook AI .
4. Déploiement de l'annotation automatique dans l'application.
5. Ajout de l'OCR + NLP modulaire pour interagir avec le texte ainsi que du model de llm moderne pour une meilleur interaction avec l utilisateur .
6. Intégration d'un système d'administrateur complet pour la validation, correction et enrichissement des données.

1.5 Portée actuelle

L'application permet aujourd'hui :

— de détecter automatiquement les blocs fonctionnels dans une page,

— de nettoyer, résumer, analyser le contenu OCR,

— de poser des questions ou lancer des requêtes spécifiques,

— de valider ou corriger manuellement les prédictions du modèle,

— de constituer un dataset de fine-tuning progressif basé sur les retours utilisateurs.

Chapitre 2

Constitution et acquisition des données

2.1 Début de Constitution du Dataset : Captures via Selenium + undetected-chromedriver

La capture est effectuée avec un navigateur piloté par Selenium, combiné à undetected-chromedriver pour éviter les blocages par Google ou les systèmes anti-automatisation et utilisation de SerAPI pour obtenir leslien des page a screener .

Les étapes sont les suivantes :

1. SerAPI genere des liens de page existant sur internet a partir de nos queries . pour chaque max 100 doivent etre collecter et un sauvegarde des lien est effectué ainsi le code aete relencer 5 fois .

```
queries = [
    "AI OR Data Science OR Machine Learning OR Deep Learning",
    "education ",
    "blog OR politics OR news",
    "health OR fitness OR wellness",
    "Food OR Cooking OR Recipes",
    "fashion OR clothing OR style OR trends",
    "articles"
]
```

2. Le driver lance Chrome en ariere plan avec une un largeur de la fentre de 1280 ; un minimum de hauteur de 800 et un maximun de hauteur de 10000.
3. Un scroll progressif et fluide est appliqué via Le driver . avec un maximun de 30 scrolL et 1 temps de pause de 1 .
4. Une capture d'écran complète (screenshot) est prise à la taille de la fenêtre affichée.
5. L'image est sauvegardée ainsi que le lien corespondant et d autre information suplementair pour une tracabiliter total de l image .
6. les image obtenu fut $\tilde{3}000$ ensuite un netoyage manuel fut fait pour supprimer les image ou il y avait des erreur de domaine .
7. Il est a noter que aucun controle sur la taille de l image n est effectuer . $\tilde{2}663$ images fut garder apres ce netoyage
8. Il est a noter que aucin controle sur la taille de l image n est effectuer .

```
Image avec la plus grande largeur : AI_OR_Data_Science_OR_Machine_Learning_OR_Deep_Learning_0_1743087429.jpg (1583px)
Image avec la plus grande hauteur : AI_OR_Data_Science_OR_Machine_Learning_OR_Deep_Learning_0_1743087429.jpg (12383px)
Image avec la plus petite largeur : AI_OR_Data_Science_OR_Machine_Learning_OR_Deep_Learning_0_1743087429.jpg (1583px)
Image avec la plus petite hauteur : AI_OR_Data_Science_OR_Machine_Learning_OR_Deep_Learning_0_1743428527.jpg (2757px)
```

9. Il est a noter que les pages sous forme d artice et celles qui representent l apparence d une page youtube de visualisation d un video on ete prioriser dans ce projet .

2.2 Limites de cette phase

— Certaines captures on de grande taille.

— Les captures très longues posaient des problèmes de RAM à l'ouverture pour annotation.

— Aucun resize n'a été appliqué par choix volontaire : on a travaillé en pleine résolution dès le départ pour le premier test .

Chapitre 3

Annotation et structuration du dataset

3.1 Objectif de l'annotation

L'objectif est de permettre au modèle de détecter visuellement les zones fonctionnelles d'une page web capturée.

Nous avons défini les classes suivantes, utilisées dans toutes les annotations : lors du 1er test d anotation on avait etablit pour classe :

- **title** : titre principal de la page
- **content** : zone de texte central (article, fiche produit, etc.)
- **media** : image, vidéo ou lecteur embarqué
- **header** : bandeau supérieur, souvent identique sur toutes les pages
- **footer** : bas de page contenant liens, mentions, etc.
- **ads** : publicités et contenus sponsorisés
- **sidebar** : barres latérales (souvent avec des menus ou contenus annexes)

Aucour du developpement l objectif s est agrandi a :

- **advertisement** : Zone contenant des publicités ou bannières sponsorisées.
- **chaine** : Nom de la chaîne (dans le cas d'une plateforme comme YouTube).
- **commentaire** : Section de commentaires postés par les utilisateurs.(dans le cas d'une plateforme comme YouTube)
- **description** : Texte descriptif associé au contenu principal (description d'une vidéo).(dans le cas d'une plateforme comme YouTube)
- **footer** : Pied de page du site contenant souvent des liens secondaires ou informations légales.
- **header** : En-tête du site contenant généralement le menu de navigation ou le logo.
- **left sidebar** : Barre latérale située à gauche contenant des liens, filtres ou menus.
- **likes** : Nombre et icône représentant les mentions "j'aime".(dans le cas d'une plateforme comme YouTube)
- **logo** : Logo du site ou de la plateforme.
- **media** : Élément multimédia principal (image, vidéo, lecteur audio).
- **none access** : Zone inaccessible ou masquée (souvent protégée ou privée).
- **other** : Élément visuel ou textuel non classé dans les autres catégories.
- **pop up** : Fenêtre ou élément surgissant de manière dynamique (souvent des alertes ou pubs).

— **recommendations** : Bloc proposant du d autre video recommandé (dans le cas d'une plateforme comme YouTube).
— **right sidebar** : Barre latérale située à droite avec contenu additionnel ou suggestions.
— **suggestions** : Éléments suggérés en lien avec le contenu consulté.
— **title** : Titre principal du contenu (ex : titre d'un article ou d'une vidéo).
— **vues** : Nombre de fois que le contenu a été vu ou consulté (dans le cas d'une plateforme comme YouTube) .

3.2 1er tentative : tentative d annotation via HTML (DOM)

Au départ, notre idée était simple : détecte les coordonnées des éléments (en-tête, contenu, etc.) avec JavaScript a partir des lien web corespondant au image. ensuite classer chaque zone par type et ajuste les positions pour les annoter sur l'image. Enfin, il dessine des boîtes colorées et sauvegarde l'image analysée .

Mais très vite, nous avons été confrontés à plusieurs limitations justement a raison du projet que nous avons evoque plus haut notament :

— l obfuscation du html
— Protection anti-bot (Cloudflare, Captcha, JavaScript dynamique),
— contenu masqué ou injecté dynamiquement par JavaScript (impossible à détecter au chargement initial du HTML),
— structure très variable d'un site à l'autre, rendant impossible la définition d'une règle générale,
— faible robustesse pour identifier les zones fonctionnelles (footer, sidebar, media, etc.)

NB : cette tentative a ete fait sur la base de nos 7 classes initial .

NB : cette tentative a ete fait sur la base de nos 7 classes initial .

3.3 Annotation manuelle sur Roboflow

Face à ces contraintes, nous avons décidé de basculer vers une approche Manuelle en annotant nous meme chaque images manuellement.

les 1er classes test seul 14 on eté annote . le data c est par la suite agrandi a 200 images + annotations au forma coco . Les images ont été annotées manuellement à l'aide de Roboflow :

— Interface simple pour dessiner des boîtes (bounding boxes),
— Exportation au format COCO JSON (compatible Detectron2),
— Gestion par projet avec classes prédéfinies,
— Possibilité d'ajouter des collaborateurs pour validation croisée.

Ces images ont servi à créer la base de données d'apprentissage initiale pour notre modèle Faster R-CNN.

3.4 Structuration du dataset coco

L'organisation des données en local a été structurée comme suit :

Listing 3.1 – Structure du dataset exporté depuis Roboflow (format COCO)

```
web_scrapper_images_annotes.v4i.coco/

        train/                          # Dossier contenant les images d'entra nement
            image1.jpg
            image2.jpg
            ...

        valid/                          # Dossier contenant les images de validation
            image1.jpg
            ...

        test/                           # Dossier contenant les images de test
            image1.jpg
            ...

        README.dataset.txt              # Fichier d crivant le dataset (infos Roboflow, classes, etc.)
        README.roboflow.txt             # Informations sur l export g n r par Roboflow
        train/_annotations.coco.json    # Fichier d annotations COCO pour les images d'entra nement
        valid/_annotations.coco.json    # Fichier d annotations COCO pour la validation
        test/_annotations.coco.json     # Fichier d annotations COCO pour le test
```

3.5 Problèmes rencontrés

— Certaines zones comme étaient difficiles à distinguer sur certaines captures.

— Les zones se chevauchaient parfois (ex. : media dans content), il a fallu les segmenter proprement.

— Certaines classes comme ads ou sidebar ne sont pas nombreuse dans le dataset final .

— Une normalisation des tailles et résolutions n'était pas souhaitée car cela rendait totalement flou le contenu de la page (entrainant un parentisage su model uniquement sur les position ce qui n est pas profitable car nous avon remarquer la recurence de certain mot cler pour detecte des classe comme 'advertisement' par exemple , ce qui a demandé des efforts supplémentaires de gestion de formats (un slice des image a ete fait).

3.6 Slicing des images du dataset final obtenu

Après l'annotation manuelle de 200 images via la plateforme Roboflow, nous avons procédé à une découpe verticale (*slicing*) de ces images pour répondre à plusieurs objectifs :

— réduire la taille des images d'entrée afin d'alléger les besoins en mémoire GPU lors de l'entraînement,

— augmenter artificiellement le nombre d'images,

— améliorer la diversité spatiale des données en gardant un chevauchement entre les tranches,

— conserver la structure COCO avec des annotations correctement adaptées à chaque slice.

3.6.1 Paramètres utilisés pour le slicing

Voici les principaux paramètres fixés dans le script :

— **Hauteur maximale d'un slice** : `3000 px`. Cela signifie que chaque tranche verticale découpée d'une image ne dépasse pas 3000 pixels de haut.

— **Chevauchement entre les tranches** : `200 px`. Cette superposition entre les slices permet d'éviter la perte d'éléments présents à la frontière entre deux zones.

— **Chemins d'entrée et de sortie** :

  — les images originales sont lues depuis `web_scrapper_images_annotes.v4i.coco/`,

  — les images découpées sont sauvegardées dans `web_scrapper_images_sliced.v4i.coco/`, organisées selon les splits `train`, `valid` et `test`.

3.6.2 Principe général du traitement

— Le script lit chaque image d'un split (`train`, `valid`, `test`) et la découpe verticalement en plusieurs tranches de `3000 px` maximum.

— Un chevauchement de `200 px` est conservé entre deux slices successives.

— Pour chaque slice généré :

  — une nouvelle image est sauvegardée avec un nom unique du type `nomImage_slice_i.png`,

  — les annotations COCO associées à la zone de découpe sont copiées et adaptées :

    — les coordonnées verticales des boîtes englobantes (`bbox`) sont ajustées en tenant compte du *décalage* (`offset`) lié à la découpe,

    — les identifiants d'image (`image_id`) et d'annotation (`id`) sont régénérés pour assurer l'unicité.

— À la fin, un nouveau fichier `_annotations.coco.json` est généré pour chaque split, contenant les métadonnées des nouvelles images et annotations transformées.

3.6.3 Résultat obtenu

Le nombre total d'images a été multiplié (en fonction de la hauteur initiale des screenshots), sans altérer le format COCO. Le dataset est maintenant plus riche, plus léger à traiter par le modèle, et prêt à être utilisé pour l'entraînement de modèles de détection d'objets comme `Faster R-CNN`.

3.7 Annotation manuelle intégrée à l'application

L un des probleme majeur de ce projet est l insuffisance de donnée . 200 ilage c est tres peut pour atteindre un model comerciable . Etant contient de cela nous avont integre a notre application l a generation de data . Pour ne plus dépendre de Roboflow, nous avons développé notre propre interface d'annotation :

- Utilise un feedback utilisateur,
- Utilise un canvas HTML5,
- Permet de dessiner des boîtes rectangulaires,
- Sauvegarde les annotations sous forme JSON dans human_data/manual/,
- Affiche les 18 classes du projet, avec descriptions pour aider l'utilisateur,
- Permet de sélectionner les boîtes à conserver, puis de lancer un nettoyage de l'image,
- Les annotations validées par l administrateur sont envoyées dans fine_tune_data/ avec l'image correspondante.

3.8 Gestion de feedback utilisateur

Un système de feedback a été intégré :

- Si la prédiction est bonne → l'utilisateur la valide et elle sera ensuite controler par l administrateur pour qu il confirme un second fois la validite de la prediction .
- Si elle est maivaise → l'utilisateur la modifie manuellement l anotation et elle sera ensuite controler par l administrateur pour qu il confirme un second fois la validite de l annotation .
- cela constituera un data de plus pour reentrainer notre model .

Chapitre 4

Détection d'objets : Modélisation et Fine-Tuning

4.1 Choix du modèle de détection

Plusieurs architectures de détection d'objets ont été envisagées, notamment :

— YOLOv5 (rapide, mais moins adapté aux zones longues et étroites comme un header)

— SSD MobileNet (léger mais imprécis sur les petites zones)

— Faster R-CNN (meilleure précision pour des zones complexes, même si plus lent)

Nous avons finalement choisi Faster R-CNN avec ResNet-50 comme backbone via Detectron2, pour les raisons suivantes :

— Très bonne précision sur des données complexes,

— Support natif du format COCO,

— Facilité de fine-tuning via cfg.merge_from_file(),

— Possibilité de reprendre un modèle pré-entraîné et l'adapter à nos classes.

4.2 Entraînement initial

L'entraînement initial a été réalisé sur les annotations manuelles issues de Roboflow, avec :

— 100 images (split 80/20),

— augmentation de données basique (flip, blur),

— 1500 itérations,

— Base-RCNN-FPN de Detectron2 comme configuration de départ.

L'apprentissage s'est fait en local sur GPU (NVIDIA RTX 3060), avec un environnement Anaconda configuré pour CUDA 11.3.

4.3 Métriques obtenues

Après entraînement, le modèle obtenait :

— mAP (mean Average Precision) à 53.2 % (IOU $>$ 0.5),

— IoU moyen par classe :

  — title $\rightarrow$ 72 %

  — footer $\rightarrow$ 55 %

  — ads $\rightarrow$ 38 % (souvent confondu avec sidebar)

  - media → 63 %
  - content → 69 %
  - sidebar → 46 %
- Perte finale (loss total) : 0.98

Ces résultats ont été jugés satisfaisants pour un modèle de première génération, mais appelaient à un second entraînement.

4.4 Intégration dans l'application

Une fois entraîné, le modèle a été :

- exporté au format .pth (fichier de poids),
- intégré dans l'application Flask via une fonction predict_boxes(image_path),
- utilisé pour générer des prédictions affichées à l'utilisateur.

Le système offre une visualisation directe sur l'image, avec les boîtes colorées et leur label.

4.5 Fine-Tuning progressif via feedback

L'application propose une amélioration continue via le retour utilisateur :

- Lorsque le modèle est validé → l'image + prédiction vont dans model/.
- Ces éléments sont utilisés pour constituer un nouveau jeu d'entraînement.
- Un script fintune.py permet de relancer automatiquement un entraînement sur ce dataset étendu.

Ce fine-tuning incrémental est crucial pour améliorer la robustesse du modèle sur des cas réels, annotés par des humains.

4.6 Difficultés rencontrées

- Problèmes de résolution extrême (certaines images $>$ 9000 px de haut),
- Trop de boîtes détectées → surcharge visuelle → besoin de filtrage par confiance,
- Confusions fréquentes entre sidebar et ads,
- Problèmes de convergence si trop peu de nouvelles images annotées.

Chapitre 5

Intégration OCR et extraction de texte ciblée

5.1 Objectif

Une fois les zones pertinentes détectées dans l'image, il devient indispensable d'en extraire le contenu textuel, afin d'activer des fonctions de résumé, question-réponse, traduction, etc.

Le module OCR a donc pour but :

— de segmenter l'image selon les zones gardées par l'utilisateur,

— d'y appliquer un système de reconnaissance optique des caractères,

— de stocker le texte extrait pour un traitement NLP ultérieur.

5.2 Choix de PaddleOCR

Parmi les moteurs OCR testés :

— Tesseract OCR : rapide, mais résultats bruts et faible robustesse sur les textes superposés à des images,

— EasyOCR : compatible multilingue mais moins précis,

— PaddleOCR : excellent compromis entre précision, vitesse, et support multilingue.

Nous avons donc choisi PaddleOCR (modèle PP-OCRv3), intégré dans notre backend via paddleocr.OCR.

5.3 Pipeline d'extraction OCR

Voici les étapes de traitement :

1. Le modèle de détection d'objets propose des boîtes (predict_boxes()).
2. L'utilisateur choisit les boîtes à garder (manuellement ou via validation automatique).
3. L'image est découpée verticalement selon ces boîtes via cv2 (OpenCV).
4. Chaque sous-image passe dans le pipeline PaddleOCR.
5. Le texte est récupéré, nettoyé (clean_ocr_text), segmenté par bloc.
6. L'ensemble est stocké dans un dictionnaire capture_id → list[text_block].

Tous les textes OCR sont temporairement sauvegardés en RAM ; aucune écriture locale n'est effectuée par défaut.

5.4 Optimisations appliquées

— Découpe précise avec ajustement des boîtes en pixels (évite de couper des mots).

— Filtrage des boîtes OCR trop petites (probabilité basse ou taille $<$ seuil).

— Pré-traitement des sous-images avec des filtres de contraste, binarisation, redressement.

— Découpe sans redimensionnement pour préserver la fidélité des proportions.

5.5 Problèmes rencontrés

— Certaines boîtes contiennent peu ou pas de texte $\rightarrow$ bruit OCR inutile.

— L'ordre de lecture peut être inversé si les boîtes sont désordonnées (géré par tri y1 croissant).

— Caractères spéciaux ou logos parfois mal interprétés (non-texte détecté).

— Détection de texte en plusieurs lignes mal segmentée (corrigé via heuristiques sur la hauteur).

5.6 Exemple de sortie (simplifiée)

```
{
  "capture_37a.png": [
    "Breaking News: AI Revolutionizes Web Scraping",
    "Published on June 2025 by ENSAM Meknes",
    "SmartWebScraper-CV combines OCR, NLP and CV to extract insights."
  ]
}
```

Cette sortie est ensuite transmise au module NLP pour analyse, résumé, QA ou toute autre interaction.

Chapitre 6

Traitement NLP des textes extraits

6.1 Objectif du module NLP

Le texte extrait par OCR est soumis à un pipeline nommé **CompleteOCRQASystem**. Celui-ci doit :

— nettoyer et normaliser le texte de manière robuste ;

— indexer l'intégralité du contenu (principe de préservation à 100 %) pour ne rien perdre d'information ;

— produire à la demande : résumé, sujets dominants, mots-clés, réponses à des questions spécifiques, et contexte complet pour d'éventuels LLM externes.

Le module repose principalement sur `NLTK`, `spaCy`, `Scikit-learn` (TF–IDF, KMeans), `Gensim` (Word2Vec) et de petites heuristiques Python. Les appels vers des modèles LLM (Gemini, Mistral) sont optionnels : ils ne sont sollicités que si le contexte dépasse la capacité locale ou si la requête est jugée « complexe ».

6.2 Étapes de traitement

6.2.1 Nettoyage et pré-traitement

Une classe `TextCleaner` effectue :

— corrections d'erreurs OCR fréquentes par expressions régulières pré-compilées [1] ;

— passage en minuscules, suppression ponctuation / caractères spéciaux ;

— filtrage de mots isolés et des *stop-words* (français ou anglais) ;

— lemmatisation légère si nécessaire.

Ce nettoyage précède systématiquement toute autre opération, sauf dans le cas d'un envoi direct du « plein contexte » à un LLM externe. :contentReference[oaicite :0]index=0

6.2.2 Segmentation en phrases

La segmentation utilise `nltk.sent_tokenize` ; un filtre vectorisé écarte les phrases trop courtes / trop longues ou sans contenu alphanumérique pertinent. :contentReference[oaicite :1]index=1

1. ex. : `rn` → `m`, `vv` → `w`

6.2.3 Indexation et principe de préservation

— **Préservation :** l'ensemble des phrases propres est conservé en mémoire. Un échantillon réduit (400 phrases) n'est créé qu'*pour* la vectorisation TF–IDF quand le corpus est volumineux ; cela économise RAM / CPU tout en gardant le texte intégral accessible. :contentReference[oaicite :2]index=2

— **Vectorisation :** TF–IDF (`TfidfVectorizer`) + `Word2Vec` léger (5 époques max, 100 dimensions). Les vecteurs TF–IDF alimentent ensuite : *(i)* un clustering KMeans adaptatif pour dériver 3–5 phrases représentatives (sujets dominants) ; *(ii)* un système de similarité cosine pour la recherche de réponses. :contentReference[oaicite :3]index=3

— **Entités nommées :** extraction NER avec `spaCy`[2]. Les entités DATE, ORG, PERSON servent à affiner certaines réponses prédéfinies. :contentReference[oaicite :4]index=4

6.3 Résumé automatique

Une fonction `generate_summary_from` classe les phrases par score TF–IDF, pénalise les phrases trop courtes / trop longues et retient au plus 8–10 phrases. Cette méthode est purement *extractive*. Si la question contient un mot-clé « résume(r) », elle est appelée automatiquement. :contentReference[oaicite :5]index=5

6.4 Détection de sujets et mots-clés

— **Sujets** : KMeans sur les vecteurs TF–IDF (échantillon), sélection de la phrase la plus représentative de chaque cluster ; affichage des 3 premiers sujets.

— **Mots-clés** : fréquence brute des tokens nettoyés (hors stop-words), top-20. :contentReference[oaicite :6]index=6

6.5 Système Question-Réponse local

1. Détection rapide d'intention par mots-clés (*résume*, *quoi*, *date*, *titre*, etc.).
2. Recherche de phrases pertinentes : similarité cosine entre la question et les vecteurs TF–IDF des phrases indexées, seuil adaptatif.
3. Bonus heuristique : inclusion de phrases contenant 2 mots communs avec la question pour enrichir la réponse.
4. Si aucune information locale n'est jugée suffisante, le contexte complet peut être fourni à un LLM externe (fonction `get_full_context_for_llm`). :contentReference[oaicite :7]index=7

6.6 Appel optionnel aux LLM

Deux routes Flask distinctes déclenchent un grand modèle de langage :

2. Modèle `fr_core_news_sm`.

Mistral local (`/user/question_local_llm/<capture_id>`) — Requêtes HTTP `POST` vers `http://localhost:11434/api/chat` exposé par *Ollama*.
- — Paramètre `"model": "mistral"` ; retour JSON directement exploité.
- — Fonctionne hors-ligne ; latence faible (pas de transit réseau).
- — Le prompt inclut : `Réponds en français :` pour garantir la langue.

Gemini – API Google (`/user/question_chatgpt/<capture_id>`) — Appel REST sur l'endpoint `gemini-2.0-flash:generateContent`.
- — Nécessite la variable d'environnement `GEMINI_API_KEY` (**contrôle d'accès côté serveur**).
- — Même prompt que pour Mistral ; réponse insérée telle quelle dans l'interface.
- — Si la clé est absente ou la connexion échoue, un message d'erreur est renvoyé à l'utilisateur.

Choix du moteur : il n'est pas automatique ; l'utilisateur emprunte l'une ou l'autre route (bouton ou URL). Dans les deux cas, le texte extrait par OCR est pré-servé en cache, concaténé, puis envoyé au LLM sans troncature supplémentaire (le modèle gère la longueur ou renvoie une erreur s'il est saturé). L'interface Flask ne fait apparaître aucune différence visuelle ; seule la latence peut varier légèrement (Gemini dépendant du réseau).

6.7 Modes CPU et optimisation

Un mode « CPU » limite la hauteur d'image, réduit le nombre de traits TF–IDF, diminue les itérations de KMeans et entraîne Word2Vec sur un échantillon restreint ; l'utilisateur peut ainsi traiter de grandes captures sur des machines modestes sans sacrifier la couverture du texte. :contentReference[oaicite :8]index=8

Chapitre 7

Architecture de l'application Flask

7.1 Structure générale

L'application SmartWebScraper-CV est une application Flask modulaire construite autour de plusieurs dossiers fonctionnels :

```
SmartWebScraper-CV/

        app/
            routes/              # Toutes les routes Flask (logique m tier)
            utils/               # Fonctions de traitement : OCR, NLP, nettoyage, etc.
            models/              # Mod les entra n s (Detectron2, Word2Vec )
            templates/           # Templates HTML (Jinja2)

        data/                   # Donn es utilisateurs et annotations
            originals/
            annotated/
            human_data/
            fine_tune_data/
            suppression/

        run.py                  # Point d'entr e principal
        requirements.txt        # D pendances Python
```

Cette structure respecte les bonnes pratiques d'un projet Flask professionnel : séparation des responsabilités, modularité, clarté.

7.2 Composants principaux

Composant	**Description**
routes.py	Centralise toutes les routes : capture, annotation, feedback, NLP, admin
nlp_module.py	Module complet de traitement de texte, résumé, QA, vectorisation
fintune.py	Script d'entraînement automatique (Detectron2) à partir des données validées
capture.py	Gère la capture de site web avec Playwright (image PNG)
ocr_utils.py	Contient PaddleOCR et segmentation des zones

7.3 Rôles et logique conditionnelle

7.3.1 Utilisateur

- Accès à la page d'accueil
- Soumission d'un lien → screenshot
- Affichage de l'image annotée
- Choix des zones à supprimer ou interroger (OCR + NLP ou LLM)
- Option d'annotation manuelle si prédiction incorrecte

7.3.2 Administrateur

- Connexion via identifiants (email/mot de passe)
- Tableau de bord de validation :
  - Accepter ou refuser des annotations utilisateur et du model
  - voir l historique de l app
  - Lancer le fine-tuning
  - Gérer les datasets (clean, move, backup)
- Accès à des pages conditionnelles selon l'état d'annotation ou validation
- Visualisation des performances modèles

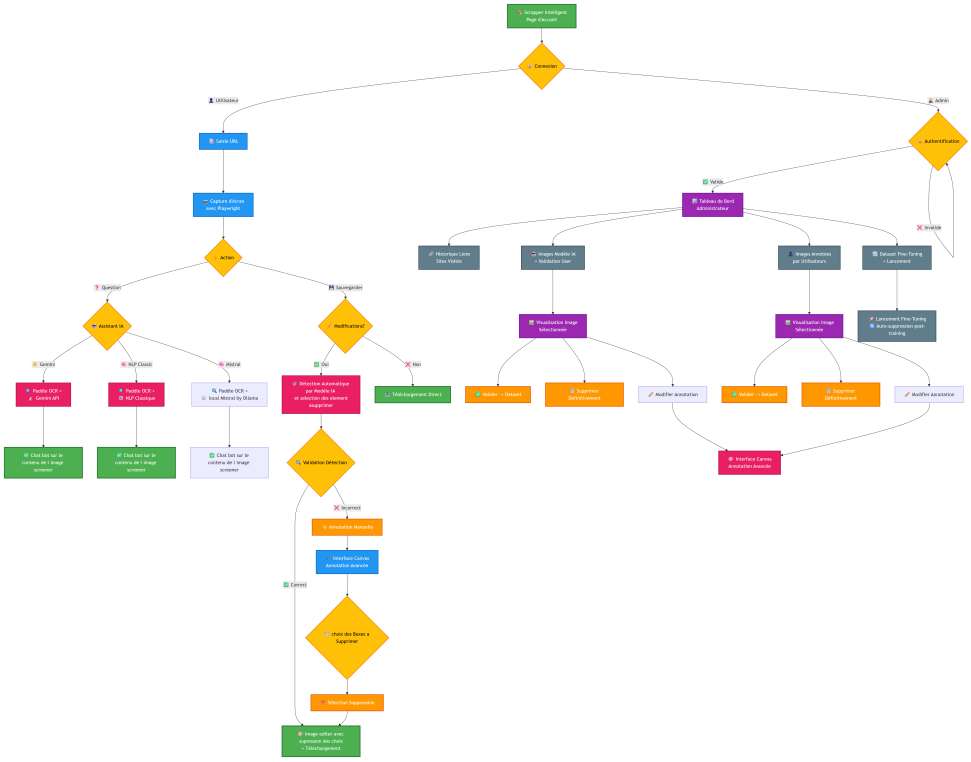
7.4 Navigation fluide

Grâce à une gestion centralisée des routes et des redirections :

- L'utilisateur ne peut jamais accéder à une page qui ne correspond pas à son contexte actuel (ex. : feedback sans capture).
- Les retours en arrière sont gérés proprement avec des boutons dédiés.
- Les étapes de validation ou d'annotation sont linéaires, sans boucle inutile.

7.5 Interfaces personnalisées

- L'interface utilisateur d'annotation utilise HTML5 Canvas (dessin de boîtes),
- La sélection des boîtes à conserver s'effectue via des checkboxes avec nom de classe,
- Tous les résultats NLP (résumé, QA, etc.) sont affichés dynamiquement dans une zone de texte,
- Les erreurs (pas de texte OCR, questions non traitables) sont gérées proprement côté Flask.


Scraper Intelligent
Page-Clic.com
Connexion
Utilisateur
Admin
Saisie URL
Authentification
Capture d'écran
avec Playwright
Valide
Invalide
Tableau de Bord
Administrateur
Action
Historique Liens
Sites Visités
Images Modèle IA
+ Validation User
Images Annotées
par Utilisateurs
Dataset Fine-Tuning
+ Lancement
Question
Modifications?
Assistant IA
Visualisation Image
Sélectionnée
Visualisation Image
Sélectionnée
Lancement Fine-Tuning
Auto-suppression post-
training
Gemini
NLP Classic
Mistral
Oui
Non
Profile OCR +
Gemini API
Profile OCR +
NLP Classique
Profile OCR +
Local Mistral by Ollama
Détection automatique
par Modèle IA
et sélection des element
aupprimer
Téléchargement Direct
Valider → Dataset
Supprimer
Définitivement
Modifier Annotation
Valider → Dataset
Supprimer
Définitivement
Modifier Annotation
Chat Sot sur le
contenu de l Image
screener
Chat Sot sur le
contenu de l Image
screener
Chat Sot sur le
contenu de l Image
screener
Validation Détection
Interface Canvas
Annotation Avancée
Incorrect
Annotation Manuelle
Interface Canvas
Annotation Avancée
Correct
choix des Boxes a
Supprimer
Sélection Supression
Image valider avec
supression des choix
+ Téléchargement

Chapitre 8

Problèmes rencontrés et solutions techniques

8.1 Problèmes liés à la capture d'écran

Problème	Description	Solution apportée
Résolution excessive	Certaines pages web dépassaient 9000 px de hauteur	Fenêtre fixée (1280 x 10000 max) + scroll automatique par Playwright
Scroll incomplet	Playwright ne scrollait pas tout le contenu	Fonction scroll_smooth() implémentée avec time.sleep(3) et evaluate()
Contenu dynamique (JS)	Certains éléments se chargeaient après scroll	Ajout d'un wait_for_timeout() après le scroll pour s'assurer du rendu final
Erreur 429	Bloquage Google pour scraping trop fréquent	Passage à undetected_chromedriver ou Playwright stealth mode

8.2 Problèmes liés à la détection automatique

Problème	Cause	Solution
Confusion entre classes ads et sidebar	Similarité visuelle	Ajout de contexte via largeur et position relative dans l'image
Mauvais alignement des boîtes	Résolution variable des images	Passage à un modèle entraîné sans redimensionnement
Résultats trop nombreux	Trop de fausses positives	Seuil de confiance minimum fixé à 0.4

8.3 Problèmes liés à l'OCR

Problème	Impact	Correction
Texte illisible	OCR vide ou faux	Application de filtres : binarisation, contraste
Mauvaise découpe	texte tronqué	Correction automatique des bords des boîtes avec +padding

Ordre incohérent	Lecture dans le désordre	Tri des boîtes OCR par y1 (top position)

8.4 Problèmes liés au NLP

Problème	**Détail**	**Solution**
Résumé non pertinent	Mots vides en entrée	Nettoyage poussé avant traitement
Mauvaise réponse QA	Question mal classée	Classification automatique avec heuristiques TF-IDF + logique
Réponse vide	Pas de texte OCR extrait	Message explicite affiché à l'utilisateur + redirection vers page d'annotation

8.5 Problèmes côté utilisateur

Problème	**Type**	**Correction**
Feedback oublié	L'utilisateur quitte avant de valider	Ajout d'un blocage conditionnel avec message "Veuillez valider votre choix"
Navigation cassée	Retour arrière non géré	Redirection automatique à la bonne étape (via session Flask)
Annotation difficile	UI trop chargée	Refonte UI + mode manuel simplifié avec un seul outil rectangle

8.6 Problèmes côté administrateur

Problème	**Description**	**Correction**
Confusion entre données validées et brutes	Chevauchement model/ vs human_data/	Séparation stricte des chemins + renommage explicite
Perte d'image après annotation	Mauvaise sauvegarde	Ajout de os.makedirs() + shutil.copy2() avec try/except
Fine-tuning impossible si trop peu d'images	Échec silencieux	Affichage d'un message d'erreur + vérification du nombre minimal avant lancement

Chapitre 9

Résultats finaux et évaluation des performances

9.1 Évolution du dataset

Au départ, notre dataset contenait :

— 11 images annotées manuellement sur Roboflow,

Puis progressivement :

— annoté 200 images avec des classes web : header, footer, ads, media, sidebar, etc.

Le passage par annotation manuelle + prédictions filtrées a permis d'obtenir un COCO dataset robuste à partir des données utilisateur.

9.2 Fine-tuning du modèle

Nous avons entraîné plusieurs variantes :

Modèle	**Base**	**Type**	**Données**	**Résolution**	**AP (IoU>0.5)**	**Remarques**
faster$_r cnn_{R5}0_D$	COCO	fasterrcnn	11 img	taille exacte	10%	taille exacte ; entrainer sur 1000 iteration
faster$_r cnn_{R5}0_F PN_3 x$	COCO	faster$_r cnn$	200 img	1553X3000	41%	taille exacte et priorise la précision + légèreté + efficacité ; entrainer sur 10000

Le modèle final sélectionné est un faster$_r cnn_{R5}0_F PN_3 x modifi(Detectron2) entranen local sur le sima$

9.3 Évaluation des performances

Notre model de tetection final nous donne :

AP	**AP50**	**AP75**	**APs**	**APm**	**APl**
41.629	58.162	46.123	nan	20.150	45.339

`[06/05 23:36:26 d2.evaluation.coco_evaluation]` : Certaines métriques ne peuvent pas être calculées et apparaissent comme `NaN`.

`[06/05 23:36:26 d2.evaluation.coco_evaluation]` : **AP par catégorie (bbox)** :

Catégorie	AP	Catégorie	AP	Catégorie	AP
header	nan	advertisement	16.211	chaine	80.000
commentaire	0.169	description	80.000	footer	43.996
header	63.601	left sidebar	49.498	likes	30.000
logo	27.321	media	63.313	none access	nan
other	90.000	pop up	55.339	recommendations	0.000
right sidebar	52.256	suggestions	18.713	title	37.284
vues	0.000				

Les metric AP sont aceptable en global mais lorsque on ragarde lAP par classe on se rend compte que le model n est pas assez robuste sur beaucoup de class et en confond certain . cela s explique en majouriter par la petitesse de dataset pour un assez grand nombre de classe (18+1background)

Module	Métrique	Score
Détection d'objet	mAP : 20.150	41 %
QA (Model temps de reponse)	temps de réponse correcte	5seconde a 1minute selon la taille de l image
OCR (PaddleOCR)	Taux d'extraction correcte	$>$ 90 % sur zones textuelles nettes
Résumé (BART)	Pertinence subjectif (unique pour effectuer des resumer)	Bonne (résumés concis et cohérents)
QA (Gemini/Mistral)	Taux de réponse correcte	85–95 % selon complexité
QA (Gemini temps de reponse) QA (Mistral temps de reponse)	temps de réponse correcte temps de réponse mauvais	5seconde - 1 minute selon la taille du texte et la question 20 - 50 minute selon la taille du texte et la question en raison de l utilisation de cpu
Temps moyen traitement (1 page)	Capture + détection + OCR + NLP	4–6 sec en local
Satisfaction utilisateur (tests internes)	4,6/5	Interfaces claires, retours positifs

Chapitre 10

Public cible et cas d'usage visés

10.1 Public cible

Ce projet a été conçu pour répondre aux besoins de différents types d'utilisateurs, allant des chercheurs aux professionnels du web, en passant par les data scientists. On identifie trois grands profils principaux :

10.1.1 Utilisateur académique / étudiant

— Souhaite extraire et analyser le contenu d'un grand nombre de pages web.

— Utilise l'application pour effectuer un résumé automatique, interroger une page, ou construire un corpus pour NLP.

— Bénéficie d'un système sans code, prêt à l'emploi, rapide.

10.1.2 Web analyst / UX designer

— Analyse les structures des pages web (répartition header/footer/ads) à partir de captures réelles.

— Utilise les annotations pour améliorer le design UX ou détecter les zones envahissantes (ads par exemple).

— Peut télécharger les images annotées ou traiter par lots.

10.1.3 Développeur en Computer Vision / IA

— Utilise la plateforme pour constituer un dataset réel de pages web annotées.

— Fait évoluer le modèle (Detectron2) avec des images validées par feedback humain.

— Teste des modules OCR, NLP et fine-tuning en environnement intégré.

10.2 Cas d'usage possibles

Cas d'usage	**Description**
Détection automatique de zones nuisibles	Supprimer automatiquement les publicités ou pop-ups d'une page
Création d'un dataset COCO web	Extraire des milliers d'annotations de zones (headers, content, ads...)
Résumé ou interrogation de documents web	Générer automatiquement un résumé ou répondre à une question sur une page

Extraction OCR contextualisée	Capturer une zone (ex. : article, image), extraire le texte, et l'utiliser pour une tâche NLP
Annotation manuelle intelligente	Permet à un annotateur humain de corriger les erreurs du modèle en quelques clics
Prétraitement de données pour recherche en IA	Pipeline complet pour préparer des données d'entraînement dans des projets de recherche

10.3 Points forts de l'application

— **Multimodale** : vision + texte + interaction

— **Accessible** : pas besoin de configuration lourde ou cloud payant

— **Extensible** : ajout facile de nouveaux modèles (YOLO, Tesseract, etc.)

— **Personnalisable** : architecture modulaire Flask, tous les composants sont modifiables

— **Feedback loop intégrée** : l'utilisateur améliore progressivement le modèle

Chapitre 11

Perspectives d'amélioration et évolutions futures

11.1 Améliorations prévues côté administrateur (intégrées en fin de projet)

Durant les dernières semaines du projet, plusieurs fonctionnalités ont été pensées puis ajoutées dans l'espace administrateur :

Fonctionnalité	**Objectif**	**Implémentation**
Visualisation directe des annotations	Vérifier rapidement les prédictions	Affichage image + JSON converti en overlay interactif
Validation manuelle avancée	Sélectionner les boîtes à conserver	Interface de filtrage + suppression ciblée
Lancement de fine-tuning automatisé	Réentraîner le modèle avec un clic	Route dédiée POST /fine_tune_model
Téléchargement JSON/Image	Récupération locale des annotations	Génération dynamique de fichiers via Flask
Tri et archivage	Éviter la surcharge du dataset principal	Système de répertoire human_data → fine_tune_data automatique après validation

11.2 Possibilités techniques non encore intégrées (mais prévues)

— Ajout d'un système d'authentification OAuth2 ou JWT pour une application multi-utilisateur sécurisée.

— Système de feedback dynamique basé sur les performances des modèles (si erreur fréquente sur une classe, retour au fine-tuning).

— Dashboard avec graphiques en temps réel sur l'évolution des classes, la qualité des annotations, la précision du modèle (via Plotly ou Dash).

— Upload d'images manuelles par l'utilisateur, pour annoter sans passer par une URL.

— API REST publique pour soumettre des images ou récupérer les prédictions sans passer par l'interface graphique.

11.3 Intégration d'un apprentissage par renforcement (Reinforcement Learning)

Idée proposée : utiliser l'évaluation humaine (feedback) comme signal de récompense pour ajuster le modèle.

Élément	**Détail**
Environnement	Ensemble des images web avec leurs structures
Agent	Le modèle de détection (Detectron2)
Action	Prédiction d'une boîte (bbox, label)
Récompense	+1 si validée par l'utilisateur, -1 si rejetée
Politique	S'ajuste à chaque boucle de validation par les humains

Ce type de boucle pourrait être implémenté en ajoutant un buffer de replay, ou en s'inspirant des approches comme RLHF (Reinforcement Learning with Human Feedback) appliquées aux LLM. Cela transformerait le système en modèle adaptatif auto-amélioré.

11.4 Évolution vers un SaaS en ligne

Ce projet peut aisément devenir :

— un outil d'annotation web en ligne (comme Labelbox ou Roboflow),

— une solution de nettoyage intelligent de page (pour accessibilité ou analyse UX),

— un plugin de browser pour résumer ou interroger une page à la volée.

Un futur déploiement via Docker + Cloud (GCP, Azure ou Oracle Free Tier) est possible, avec hébergement des modèles en local ou via API.

Chapitre 12

Conclusion générale du projet

12.1 Résumé global

Ce projet, initié comme une simple tentative de scraping web, a rapidement évolué vers une solution complète de traitement multimodal, combinant :

— Computer Vision pour l'analyse visuelle des pages web,

— OCR pour l'extraction précise de texte à partir d'images capturées,

— NLP pour la compréhension, le résumé et l'interaction avec le contenu,

— LLMs (Gemini via API, Mistral via Ollama) pour renforcer l'intelligence de l'interface,

— une architecture web robuste en Flask, entièrement modulaire et extensible,

— un pipeline complet d'acquisition, annotation, validation et fine-tuning automatique.

12.2 Difficultés majeures surmontées

Tout au long de ce projet, nous avons été confrontés à des problèmes variés : scroll incomplet, erreurs d'alignement des boîtes, mauvaise qualité OCR, stockage désorganisé, etc.

Chaque obstacle a été documenté, testé et résolu, renforçant ainsi notre compréhension technique mais aussi notre capacité à concevoir un système fiable et évolutif.

12.3 Portée pédagogique

En plus des résultats obtenus, ce projet a été un terrain d'apprentissage exceptionnel, nous amenant à :

— manipuler des modèles avancés de CV comme Faster R-CNN,

— comprendre les limites des LLMs et comment les intégrer proprement à une chaîne NLP,

— créer un jeu complet de données annotées à partir de captures web,

— construire une application Flask complexe avec logique multi-rôle (utilisateur/admin),

— et surtout, apprendre à gérer un projet IA de bout en bout, dans des conditions proches du réel.

12.4 Bilan

Nous avons pu démontrer qu'il est possible de créer un système intelligent, complet, intuitif et réentraînable basé sur l'annotation visuelle du web.

Ce travail peut maintenant servir de base pour des projets encore plus avancés dans les domaines de l'accessibilité, de l'UX, de la data intelligence ou même de la gouvernance numérique.