

RAPPORT DE STAGE

En vue de l'obtention du diplôme de Master professionnel en

Business Analytics

Soutenu le : Septembre 2025

Réalisé par : **Saadaoui Mayssa**

« **Automatisation de la saisie des factures** »

17/02/2025 – 17/06/2025

Maître de stage : M. Nizar Jilani

Encadrant académique : M. belhassen meftahi

Année universitaire 2024/2025

	Nom & Prénom	Signature
Maître de stage	M. Nizar Jilani	
Encadrant académique	M. Belhassen Meftahi	

Résumé :

Ce rapport présente le travail effectué dans le cadre d'un stage de fin d'études pour le Master professionnel en Business Analytics. Le projet porte sur l'**automatisation de la saisie des factures**, un processus encore largement manuel dans de nombreuses entreprises.

L'objectif principal est de concevoir une solution capable d'extraire automatiquement les informations pertinentes à partir d'une facture scannée : **numéro, date, montant, fournisseur, etc.** La démarche repose sur l'utilisation combinée de techniques d'intelligence artificielle, de Deep Learning pour la détection d'entités visuelles, et de reconnaissance optique de caractères (OCR) à l'aide de Tesseract.

Le projet a permis d'automatiser une grande partie de la chaîne de traitement, réduisant ainsi les erreurs humaines et le temps de saisie. Il ouvre également la voie à des évolutions futures vers une automatisation complète du traitement documentaire administratif.

Mots-clés : automatisation, factures, OCR, Deep Learning, traitement documentaire

Abstract :

This report presents the work carried out as part of a final internship for the Master's degree in Business Analytics. The project focuses on automating invoice data entry, a process still largely manual in many companies.

The main goal is to design a solution that can automatically extract relevant information from a scanned invoice (number, date, amount, supplier, etc.).

The approach combines artificial intelligence techniques, deep learning for visual entity detection, and optical character recognition (OCR) using Tesseract.

The project successfully automated much of the processing chain, reducing human errors and input time. It also paves the way for future developments toward full automation of administrative document processing.

Keywords: automation, invoices, OCR, Deep Learning, document processing

Dédicaces

Avec toute ma reconnaissance, je dédie ce mémoire à :

Mes chers parents,

Pour votre tendresse inépuisable, votre soutien sans faille et vos innombrables sacrifices depuis le premier jour de ma vie. Les mots me manquent pour exprimer la fierté, la gratitude et l'amour profond que je vous porte. Ce travail n'est que le reflet de vos efforts silencieux, de votre patience et de votre présence constante. Que ce succès soit le fruit de toutes ces années de dévouement. Merci pour tout. Que Dieu vous protège et vous garde toujours.

Mes sœurs,

Merci pour votre présence, vos encouragements et votre affection tout au long de mon parcours. Vous avez été des piliers de motivation et de réconfort. Je vous souhaite à mon tour tout le bonheur, la réussite et l'épanouissement que vous méritez.

Mes amis les plus chers,

Dans la vie, il est rare de croiser des personnes prêtes à vous tendre la main au bon moment, sans rien attendre en retour. J'ai eu la chance d'en avoir autour de moi. Merci pour vos encouragements, vos mots justes, vos sourires réconfortants et cette précieuse amitié. Vous avez contribué à rendre ce chemin plus doux et plus riche.

À toutes celles et ceux que j'aime et qui m'aiment,

J'espère avoir été à la hauteur de vos espérances et que ce travail témoigne de toute ma gratitude. Vous êtes les sources invisibles de ma force, mes amis, mes sœurs de cœur et d'esprit.

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui m'ont accompagnée et soutenue tout au long de ce projet de fin d'études.

Je remercie tout particulièrement **M. Belhassen Meftahi**, mon encadrant académique, pour son suivi rigoureux, ses conseils avisés et sa bienveillance durant toutes les étapes de ce travail. Sa disponibilité et son expertise m'ont grandement aidée à structurer et à mener à bien ce mémoire, tout en approfondissant mes connaissances dans le domaine de la Business Analytics.

Mes remerciements vont également à **M. Nizar Jilani**, mon maître de stage, pour m'avoir accueillie au sein de son équipe et pour m'avoir confié un projet aussi stimulant que celui de l'automatisation de la saisie des factures. Son encadrement professionnel, sa confiance et ses orientations techniques m'ont permis de progresser considérablement, tant sur le plan technique que personnel.

Je remercie aussi les **membres du jury** pour le temps qu'ils consacrent à l'évaluation de ce travail, ainsi que pour leurs retours enrichissants.

Enfin, je souhaite adresser ma reconnaissance à l'ensemble des **enseignants de l'École Supérieure Privée de Management de Tunis – ESPRIT School of Business**, pour la qualité de leur enseignement, leur accompagnement constant et l'environnement d'apprentissage stimulant offert tout au long de mon cursus.

Table des matières

CHAPITRE 1 :	1
CADRE GENERAL DU PROJET	1
Introduction	2
1 Présentation de l'organisme d'accueil	2
1.1 Présentation de la société	2
1.2 Organigramme de la société	3
1.3 Présentation du logiciel OMRAN Estate	4
1.4 Contexte du projet	5
2 Cadre du projet:	6
2.1 Contexte	6
2.2 Analyse de l'existant	6
2.2.1 Étude de l'existant	7
2.2.2 Critique de l'existant	8
2.2.3 Solution envisagée:	9
2.2.4 Définition des besoins fonctionnels et non fonctionnels	10
3 Les approches méthodologiques	12
3.1 KDD :	12
3.2. SEMMA	13
3.3. CRISP-DM :	14
3.4. Avantages et inconvénients des méthodologies	16
3.5 Méthodologie Agile Scrum:	17
4 Méthodologie adoptée : CRISP-DM	18
4.1 Mise en œuvre concrète	19
4.1.1 Compréhension du problème métier	20
4.1.2 Compréhension des données	20
4.1.3 Préparation des données	21
4.1.4 Modélisation	21
4.1.5 Évaluation	22
4.1.6 Déploiement	22
4.2 Conclusion sur le choix	22
5 Environnement logiciel et Technique:	23
5.1 Environnement logiciel	23
5.1.1 Logiciels utilisés	23
5.1.2 Bibliothèques utilisées	24
5.1.3 Langages utilisés	25
5.2. Équipement matériel	26
Conclusion	26

CHAPITRE 2	27
-------------------	-----------

ETUDE DE L'ETAT DE L'ART ET FONDEMENTS THEORIQUES	27
--	-----------

Introduction	28
---------------------	-----------

1	Introduction à l'intelligence artificielle	28
1.1.	Définition de l'intelligence artificielle et ses sous-domaines	28
1.2.	Les types d'apprentissage en IA	28
1.3.	L'émergence du Deep Learning	29
2	Familles de modèles d'apprentissage profond	30
2.1.	Réseaux de neurones artificiels (ANN)	30
2.2	Réseaux convolutifs (CNN)	31
2.3.	Réseaux récurrents (RNN) et variantes	31
2.5.	Traitement du langage naturel (NLP) et LLM	34
2.5.1.	Traitement automatique du langage naturel (NLP)	34
2.5.2.	Tâches classiques du NLP	35
2.5.3.	Modèles de langage de grande taille (LLM)	35
**	Focus sur Gemini	37
**	Focus sur Gemma	37
2.6.	Vision par ordinateur (Computer Vision)	38
2.6.1.	Objectifs et enjeux	38
2.6.2	Object Detection	38
2.6.3.	Fonctionnement technique des bibliothèques OCR	48
2.6.4.	Limites et améliorations	48
3	.Développement d'une application Web	49
3.1	un framework moderne pour la création d'API	49
3.1.1	Alternatives à FastAPI : Flask et Streamlit	49
3.2	Interface utilisateur basée sur Next.js	50

Conclusion de chapitre:	54
--------------------------------	-----------

CHAPITRE 3	55
-------------------	-----------

ARCHITECTURE DE LA SOLUTION ENVISAGEE	55
--	-----------

Introduction:	56
----------------------	-----------

1	Architecture Globale de la Solution	56
---	-------------------------------------	----

Section1: Compréhension et préparation des données (Data Understanding & Data Preparation)	57
---	-----------

1.	Compréhension des données (Data Understanding)	58
1.1	Origine et nature des données	58
1.2	Objectifs de l'analyse exploratoire	58
1.3	Analyse qualitative et statistique	58
1.4	Diversité structurelle et linguistique	59
1.5	Contraintes spécifiques rencontrées	59
1.6	Conclusion	59
2.	Préparation des données	60

2.1 Normalisation des formats de documents	60
2.2 Amélioration de la Qualité des Images par Prétraitement	61
2.4 Vérification de la qualité -conversion :	64
2.5 Annotation des données pour la détection d'objets (YOLOv8)	65
Mise en pratique	66
Section 2 Entraînement et exécution du pipeline d'analyse	72
1. Module de Classification des Documents (Facture vs Non-Facture)	72
1.2 Technologies et Méthodes Utilisées	73
1.3 Logique de Classification	73
1.4 Processus de décision	74
1.5 Architecture du Module de Classification	77
2. Module de Détection d'Entités Visuelles dans les Documents avec YOLOv8	77
2.1. Détection des Logos et des Cachets	78
2.2 Objectif et Enjeux	78
2.3 Entraînement avec YOLOv8	78
1.5 Vitesse et efficacité	86
2. Détection de Signatures : mise en œuvre et évaluation	90
2.1 Intégration et chargement du modèle	90
2.2 Métriques de performance du modèle YOLOv8 pour la détection de signatures: Ces métriques (précision, rappel, mAP) sont issues de la page officielle du modèle yolov8s- signature-detector sur HuggingFace, présentée par tech4humans. Elles ont été calculées sur le jeu de test fourni par les auteurs du modèle :	91
2.3 Inférence avec le modèle de détection de signatures	92
3. Module de TEXT EXTRACTION	93
3.1 Fonctionnement de l'Extractor OCR	95
3.2 Score de Confiance OCR	96
4. Analyse Sémantique IA : Gemini API et Gemma Local	96
4.1 Analyse avec Gemini Flash 1.5 (API Google AI Studio)	96
4.2 Étapes d'utilisation	97
4.3 Score de Confiance (Gemini)	97
4.4.1 Description	98
4.4.2 Étapes d'intégration dans le backend	98
4.4.3 Score de Confiance (Gemma)	98
5 Mécanisme de Fallback	99
5.1 Fonctionnement dans le code	99
Conclusion du Chapitre 3 : Architecture de la solution envisagée	100
CHAPITRE 4	101
IMPLEMENTATION TECHNIQUE, DEPLOIEMENT ET INTERFACE UTILISATEUR	101
Introduction	102
1 Architecture logicielle de la solution	102
2. Implémentation avec FastAPI	103
2.1 Documentation Interactive avec Swagger	103

2.2 Endpoints REST:	104
	111
3. Déploiement du Backend avec Docker	111
3.1 Construction de l'image Docker	111
3.2 Orchestration avec Docker Compose	112
3.3 Étapes d'exécution	112
3.4 Concepts Clés à retenir	113
4. Interface Web avec Next.js	114
4.1 Fonctionnalités principales	114
4.2 Intégration avec le Backend	116
4.3 Technologies et structure technique	116
4.4 Limites observées	117
4.5 Pistes d'amélioration	118
4.6 Conclusion	118
5. Intégration continue et modularité	118
Conclusion	118
CONCLUSION GENERALE	120
BIBLIOGRAPHIE	ERREUR ! SIGNET NON DEFINI.

Liste des tableaux

Tableau 1: Avantages et inconvénients des méthodologies	16
Tableau 2: les logiciels utilisés.....	23
Tableau 3: Bibliothèques utilisées	24
Tableau 4: Langages utilisés	25
Tableau 5: Comparaison entre SSD et YOLO	47
Tableau 6: Comparaison entre flask ,streamlit et fast api.....	49
Tableau 7: Avantages de Docker	53
Tableau 8 :statistiques	58
Tableau 9:contraintes	59
Tableau 10: jeu de données	66
Tableau 11: Les principaux hyperparamètres	80
Tableau 12: structure de Yolo	84
Tableau 13: résultat	85
Tableau 14: un extrait des résultats obtenus	87
Tableau 15: Métrique	91
Tableau 16: Résultat	92
Tableau 17mécanisme de fallback.....	99
Tableau 18récap.....	99
Tableau 19: tableau récapitulatif	105
Tableau 20: récapitulatif	113
Tableau 21: relation	116

Liste des figures

Figure 1:principaux clients et partenaires	3
Figure 2 : Organigramme de Yasmine Engineering Systems	4
Figure 3:Vue d'ensemble du logiciel OMRAN Estate — source : site officiel.....	4
Figure 4: Modules fonctionnels du logiciel OMRAN Estate (extrait du site officiel).....	5
Figure 5:Interface du module Trésorerie du logiciel OMRAN Estate.....	8
Figure 6:Processus KDD.....	13
Figure 7:: Processus KDD	14
Figure 8:Processus CRISP-DM	15
Figure 9:Cycle Scrum illustrant les Sprints, le Product Backlog et la Rétrospective	18
Figure 10: Les différentes phases de la méthodologie CRISP-DM	21
Figure 11: Hiérarchie conceptuelle de l'intelligence artificielle et de ses sous-domaines	29
Figure 12: Architecture typique d'un réseau de neurones artificiel (ANN)	30
Figure 13:CNN Architecture	31
Figure 14: Architectures du Transformer, de GPT et de BERT	33
Figure 15: architecture Faster R-CNN.....	41
Figure 16: Architecture du modèle SSD (Single Shot MultiBox Detector)	43
Figure 17: Architecture de YOLOAvantages de YOLO	45
Figure 18: interface de V0.....	51
Figure 19 : Interface de Spaces pour ai app directory dans Hugging face.....	51
Figure 20: Schéma de l'architecture globale de la solution pipeline.....	56
Figure 21: les bibliothèques	60
Figure 22: capture d'écran illustrant le prétraitement	61
Figure 23: validation ,vérification, conversion en gris des images	62
Figure 24: filtrage bilatéral.....	63
Figure 25: Script python de Binarisation Adaptative.....	63
Figure 26: – Script Python de regroupement automatique des images en lots de 500 ...	64
Figure 27: Script Python de séparation et filtrage automatique des données pour l'entraînement et la validation.	68
Figure 28:Exemple de répertoire 'train' du dataset YOLOv8 contenant des factures annotées.	68
Figure 29: Annotation manuelle avec Labellmg sur une facture : encadrement du logo (classe 0) et du cachet (classe 1).	69
Figure 30: – Exemple de fichier d'annotation YOLOv8 généré après annotation manuelle.	70
Figure 31:– Exemple de fichier d'annotation YOLOv8 généré après annotation manuelle.	70
Figure 32: fichier data.yamll	71
Figure 33:fichier data.yamll	77

Figure 34:les bibliothèques nécessaires pr YOLO.....	79
Figure 35:fichier data.yaml.....	79
Figure 36: : entraînement de modèle Yolo sur 100 epochs	82
Figure 37: les couches de modèle Yolo*	83
Figure 38: script python pour l'évaluation	85
Figure 39: script python pour prédire.....	86
Figure 40: pipeline d'inférence	88
Figure 41::Exemple de facture annotée avec détection de logo, cachet et métadonnées OCR	90
Figure 42:script python de chargement de modèle d'après Hugging Face.	90
Figure 43: exemple de facture testée.....	92
Figure 44:pipeline.....	94
Figure 45:Flux HTTP typique du traitement documentaire intelligent.....	103
Figure 46:Capture à insérer : page Swagger UI montrant les endpoints (/docs)	104
Figure 47:Interface Swagger – POST/extract -invoice with Gemini	105
Figure 48:Swagger – POST /extract-invoice (avec exemple de réponse)	106
Figure 49 : réponse JSON – Gemma structuration locale.....	107
Figure 50: réponse JSON – Gemma structuration locale	107
Figure 51:JSON avec texte brut OCR	108
Figure 52:Interface Swagger – Endpoint POST /verify-invoice pour la vérification automatique d'un document	109
Figure 53:Swagger /verify-invoice avec booléen is_invoice: true	109
Figure 54:Interface SWAGGER : extract-entities	110
Figure 55:Interface SWAGGER: extract-entities-with-ocr	111
Figure 56: aperçu du fichier Dockerfile dans l'éditeur de code.....	112
Figure 57: fichier docker compose.yml	112
Figure 58: Terminal en cours d'exécution (docker-compose up) avec API démarrée	113
Figure 59:Captures insérées ci-dessous de l'interface web	115
Figure 60: structure de l'interface utilisateur	117

Introduction Générale

« Dans un contexte marqué par le développement rapide des technologies numériques et l'expansion constante des activités des entreprises, l'automatisation appuyée par l'intelligence artificielle et le Deep Learning s'impose comme une solution incontournable. Elle permet de réduire le temps consacré aux tâches répétitives, de fiabiliser les données traitées et d'optimiser l'utilisation des ressources, tout en renforçant la performance et la compétitivité des organisations.

La saisie manuelle de documents administratifs et financiers, notamment les factures fournisseurs, demeure aujourd'hui l'une des tâches les plus chronophages et exposées aux erreurs. La vérification, l'extraction et l'enregistrement de ces informations stratégiques nécessitent encore une intervention humaine à chaque étape, ce qui entraîne une perte de temps et un risque d'inexactitudes ou de non-conformité.

À travers sa plateforme "OMRAN Estate", **Yasmine Engineering Systems** s'inscrit pleinement dans cette dynamique de modernisation.

"OMRAN Estate" est une solution logicielle globale, modulaire et intégrée, conçue pour couvrir l'ensemble du cycle de vie d'un projet immobilier, depuis l'analyse de faisabilité jusqu'à la commercialisation et le suivi après-vente.

Ses modules interconnectés (Gestion Administrative, Patrimoine, Achats, Trésorerie, Projets, Ventes, Location, Service Après-Vente) offrent aux utilisateurs une vision centralisée et cohérente des informations opérationnelles.

Le module Trésorerie, au cœur de ce projet, assure le suivi des flux financiers et le traitement des factures fournisseurs. Actuellement, ce processus reste manuel, générant une perte de temps, une surcharge de travail et un risque d'erreurs à grande échelle.

Face à ces constats, le présent Projet de Fin d'Études a pour objectif de concevoir et de mettre en place une solution automatisée et intelligente dédiée à la gestion des factures dans le module Trésorerie, afin de réduire la saisie manuelle, gagner en précision et améliorer l'efficacité du processus. »

Ce rapport est structuré en cinq chapitres qui présentent le déroulement du projet de manière progressive :

-
- Le **chapitre 1**, intitulé « *Contexte général* », présente l'organisme d'accueil, le cadre du projet, une analyse de l'existant ainsi que la solution proposée. Il introduit également la méthodologie de gestion de projet adoptée et l'environnement technique mobilisé.
 - Le **chapitre 2** est consacré aux *fondements théoriques* du projet. Il détaille les concepts clés sur lesquels repose la solution, notamment l'OCR, la vision par ordinateur, et le NLP.
 - Le **chapitre 3** présente la conception de la solution, incluant l'architecture du système, la préparation des données, et les choix technologiques pour le traitement OCR, la détection d'objets et l'analyse sémantique.
 - ☐ Le **chapitre 4** décrit l'implémentation, le déploiement du backend avec Docker, et l'interface web développée en Next.js, en illustrant les interactions entre l'utilisateur et les différents modules du système.

Chapitre 1 :

Cadre général du projet

Introduction

Ce chapitre est consacré à la présentation du cadre général de notre projet. Tout d'abord, nous Présentons l'organisme d'accueil « YASMINE ENGINEERING SYSTEMS », puis nous effectuons une analyse de l'existant. Il s'agit d'une étape primordiale qui permet de dégager la solution adéquate et appropriée aux différentes anomalies soulevés.

1 Présentation de l'organisme d'accueil

1.1 Présentation de la société



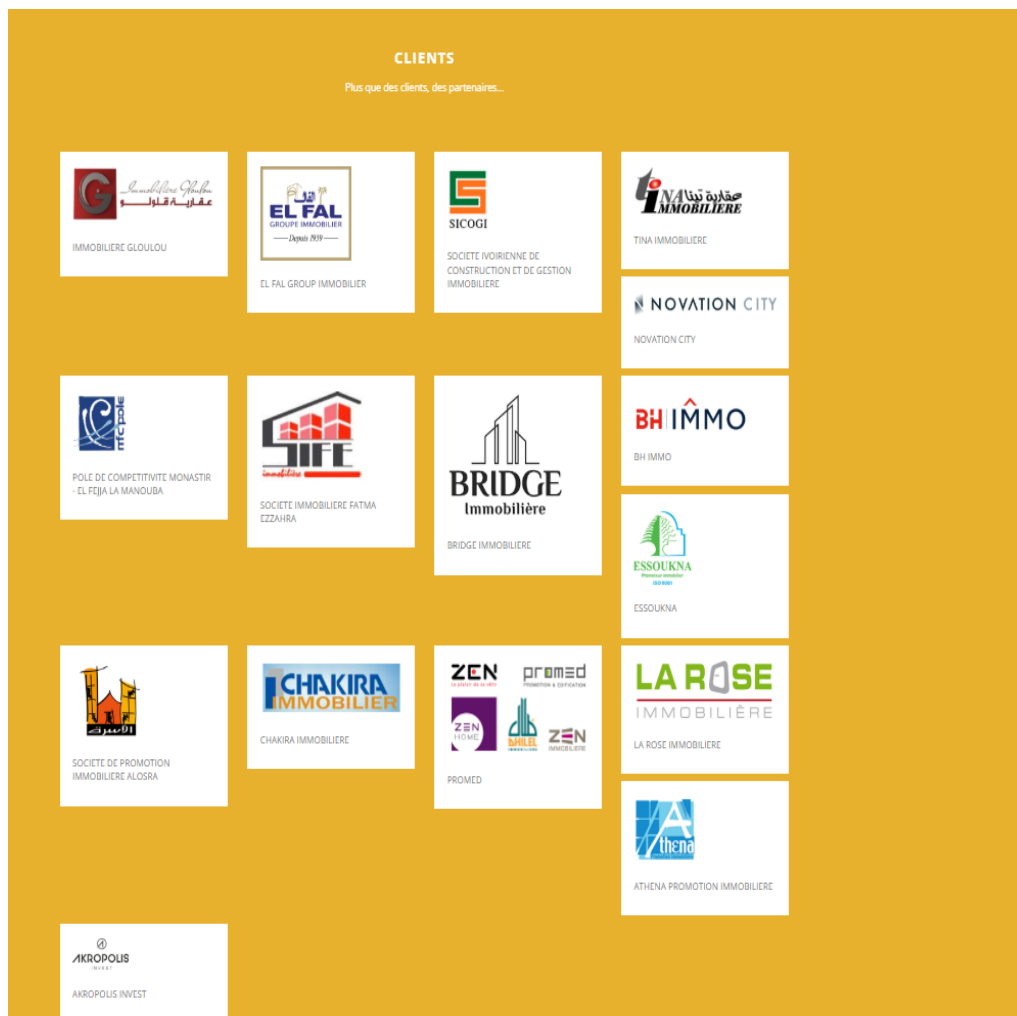
« Yasmine Engineering Systemes» est une société privée d'ingénierie logicielle, fondée en 2005 et implantée au Pôle Technologique El Ghazela à Tunis. Elle est spécialisée dans la création, la diffusion et l'intégration de solutions technologiques innovantes, spécifiquement conçues pour le secteur de l'immobilier.

L'entreprise propose une large gamme de prestations : conception de logiciels sur mesure, intégration de systèmes d'information, conseil et mise en place de stratégies organisationnelles adaptées. Grâce à une équipe pluridisciplinaire composée de techniciens, développeurs, chefs de projet et experts métier « Yasmine Engineering Systemes» accompagne ses clients dans leur transition numérique en s'appuyant sur des technologies modernes et fiables

Depuis sa création , l'entreprise a su établir une relation de confiance avec un portefeuille varié de promoteurs immobiliers et de partenaires de premier plan, répartis sur plusieurs régions et secteurs, en Tunisie comme à l'étranger. Cette fidélité témoigne de la crédibilité et du sérieux de la société sur le marché.

La figure suivante présente quelques-uns de ses principaux clients et partenaires :

✓



✓

Figure 1: principaux clients et partenaires

1.2 Organigramme de la société

«Yasmine Engineering Systems » dispose d'une structure organisationnelle claire, articulée autour de quatre unités majeures : développement logiciel, support technique, recherche & développement et service commercial.

Cette organisation favorise la synergie entre les pôles et garantit une réponse optimale aux besoins des clients.

Notre stage de fin d'études s'est déroulé au sein du département Recherche et Développement, moteur de l'innovation et de l'amélioration continue des solutions logicielles de l'entreprise.

L'organigramme suivant illustre cette structure :

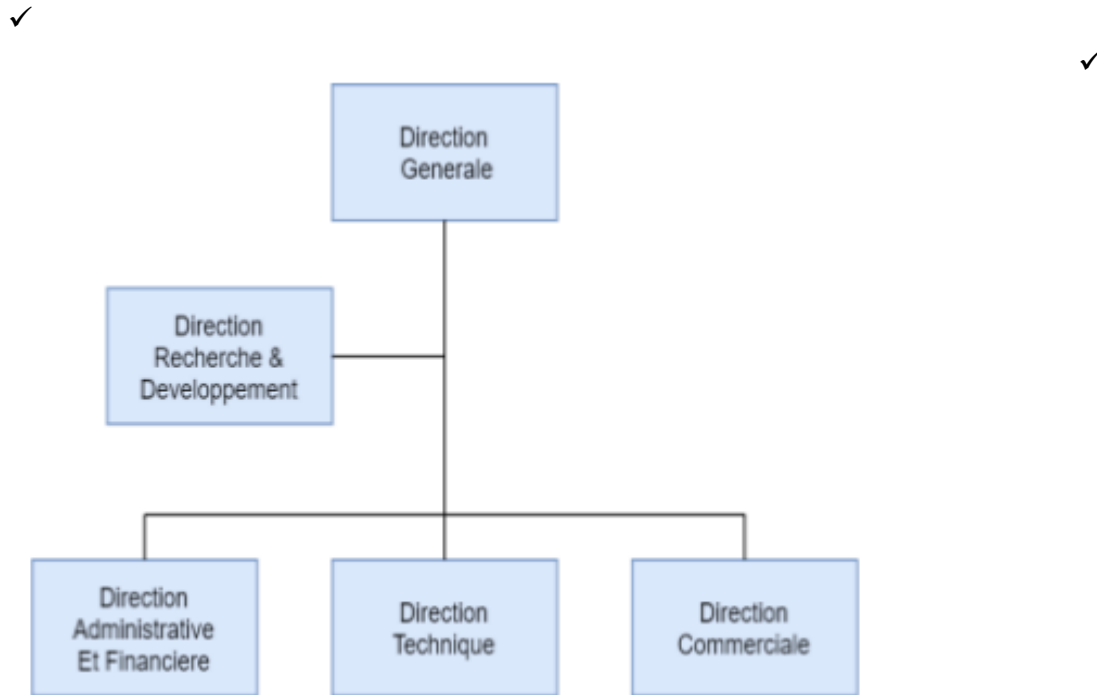


Figure 2 : Organigramme de Yasmine Engineering Systems

1.3 Présentation du logiciel OMRAN Estate

Au cœur de ses services, **OMRAN Estate** est le produit phare de «Yasmine Engineering Systems » C'est une solution **modulaire, ergonomique et intuitive**, conçue pour prendre en charge l'ensemble du cycle de vie d'un projet immobilier : depuis l'étude d'opportunité jusqu'à la réalisation, la commercialisation et le suivi après-vente.

Ce logiciel centralise toutes les données métier dans une source unique et sécurisée, permettant aux utilisateurs de piloter leurs opérations quotidiennes de manière fiable, tout en masquant la complexité technique sous-jacente.



Figure 3: Vue d'ensemble du logiciel OMRAN Estate — source : site officiel

OMRAN Estate s'organise autour de plusieurs modules principaux, interconnectés pour garantir la cohérence des processus et des informations : **Administration, Patrimoine, Trésorerie, Achat, Projet, Vente, Location, Service Après-Vente.**

Cette architecture modulaire offre une couverture fonctionnelle complète, permettant aux promoteurs immobiliers de gérer efficacement leurs activités et de répondre aux exigences du secteur.

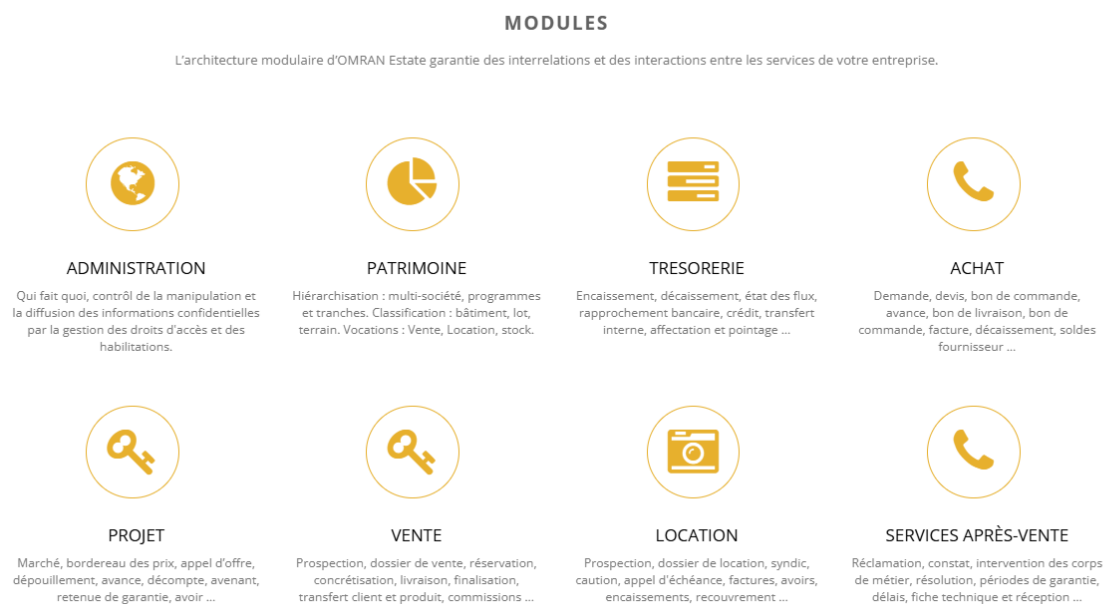


Figure 4: Modules fonctionnels du logiciel OMRAN Estate (extrait du site officiel)

1.4 Contexte du projet

Dans le cadre de sa démarche d'amélioration continue, «Yasmine Engineering Systems » a identifié certaines limites majeures dans le traitement actuel des factures fournisseurs, notamment au sein du module Trésorerie du logiciel **OMRAN Estate**. Bien que ce module soit performant et complet pour le suivi des flux financiers, la saisie des factures reste encore largement manuelle, ce qui représente une tâche chronophage et source de multiples erreurs.

Actuellement, chaque facture reçue doit être traitée étape par étape :

- Un opérateur saisit manuellement dans le système toutes les informations clés : référence de la facture, date, fournisseur, montant HT, TVA, montant TTC, nature de la dépense, projet ou rubrique budgétaire associée.

- Ensuite, le même opérateur doit vérifier la cohérence des données, souvent en comparant plusieurs pièces justificatives.
- Une fois la saisie validée, un document scanné de la facture est importé et attaché via le bouton « Attachement » du module.
- Le logiciel permet de joindre différents formats (PNG, JPEG...), ce qui, en théorie, facilite le stockage, mais multiplie aussi les risques : il n'est pas rare que des pièces non pertinentes soient importées (quittances, tickets restaurant, reçus de chèques, bons de livraison, etc.).

Avec le volume croissant de factures, cette méthode génère des pertes de temps, des doublons et des erreurs de saisie, ce qui nuit à la fiabilité des informations financières et surcharge inutilement le personnel.

2 Cadre du projet:

2.1 Contexte

Dans le but d'obtenir mon "Master Professionnel en Business Analytics" à ESPRIT SCHOOL OF BUSINESS " et de mettre en pratique les connaissances acquises au cours des quatre derniers semestres, j'ai effectué un stage de fin d'études de 5 mois au sein de l'entreprise « Yasmine Engineering Systems », spécialisée dans l'édition de solutions logicielles pour la gestion des projets immobiliers.

Au cours de ce stage, j'ai travaillé sur le développement d'une application intelligente permettant d'automatiser la saisie et l'extraction des informations des factures fournisseurs dans le module Trésorerie du logiciel "**OMRAN Estate.**"

2.2 Analyse de l'existant

L'analyse de l'existant a pour objectif de comprendre en détail le fonctionnement actuel du module Trésorerie utilisé par « Yasmine Engineering Systems » afin d'identifier les limites du processus de traitement des factures fournisseurs et de justifier la mise en place d'une solution automatisée.

2.2.1 Étude de l'existant

OMRAN Estate, solution développée par «Yasmine Engineering Systems » est un logiciel de gestion immobilière modulaire qui centralise toutes les données liées à un projet immobilier, depuis l'étude initiale jusqu'au suivi après-vente.

Parmi ses modules clés, le **module Trésorerie** joue un rôle essentiel pour le suivi des encaissements, des décaissements et de la gestion comptable des factures fournisseurs.

Actuellement, le traitement des factures se fait de manière entièrement manuelle. Le processus standard est le suivant :

- Un opérateur administratif reçoit la facture sous format papier ou image.
- Il ouvre le formulaire de saisie du module Trésorerie.
- Il saisit manuellement toutes les informations : numéro de la facture, date d'émission, fournisseur, projet concerné, rubrique budgétaire, nature de la dépense, montant HT, TVA, montant TTC.
- Enfin, il importe et rattache le fichier scanné de la facture via la fonction Attachement.

Dans la pratique, ce flux de traitement permet de centraliser et d'archiver les factures, mais il reste entièrement dépendant de la saisie humaine et du classement manuel.

Pour illustrer cela, la figure suivante présente un **exemple d'interface du module Trésorerie**, avec les flux financiers réels et prévisionnels :

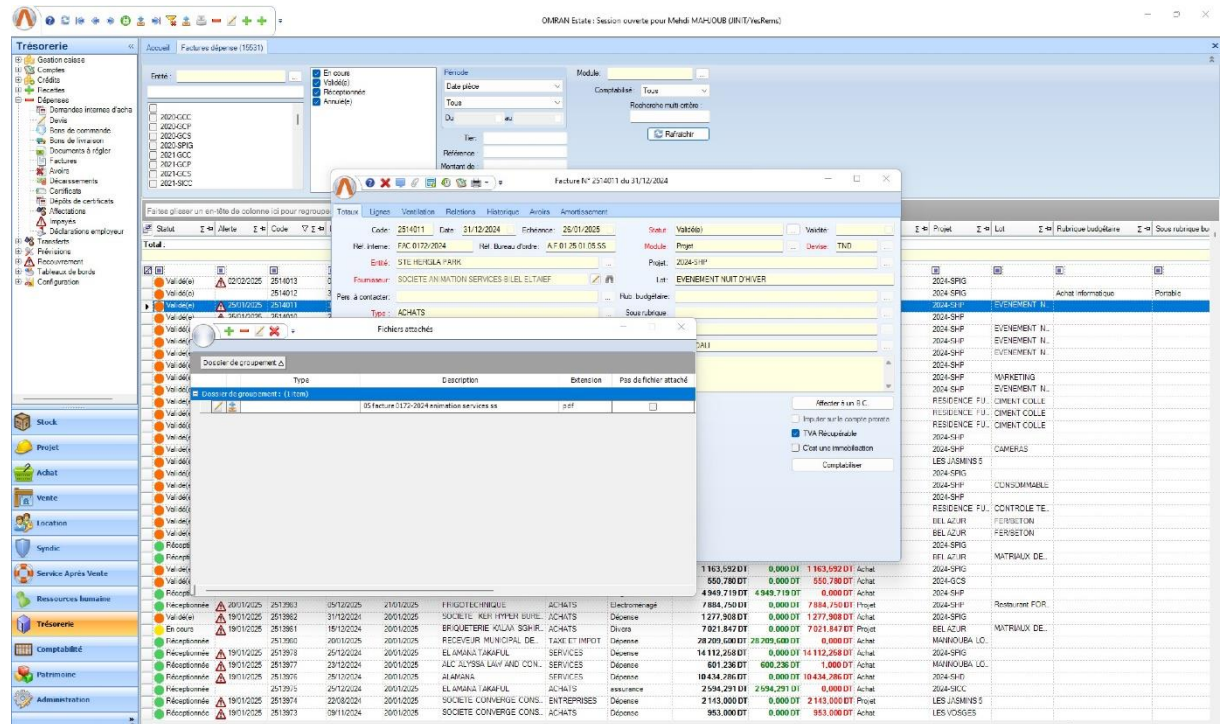


Figure 5: Interface du module Trésorerie du logiciel OMRAN Estate

2.2.2 Critique de l'existant

L'examen approfondi de la procédure met en évidence plusieurs contraintes significatives:

✓ Charge manuelle élevée :

Il est nécessaire d'entrer chaque facture champ par champ, une opération qui consomme du temps et mobilise un employé pour une activité répétitive.

✓ **Répétitivité non bénéfique** : La répétition de ces tâches nuit à la productivité des équipes, qui pourraient plutôt se consacrer à des opérations d'analyse ou de contrôle ayant une valeur ajoutée plus élevée.

✓ **Risque d'erreurs** : L'entrée manuelle amplifie le risque d'erreurs : fautes de numéros, omission de champs, répétitions ou incohérences qui compromettent la précision des données comptables.

✓ Classement imprécis :

La fonction d'attachement ne vérifie pas automatiquement le contenu : des pièces non pertinentes peuvent être ajoutées à la place de la facture réelle, entraînant des erreurs de classement et une perte de temps lors des contrôles ultérieurs.

Au regard de ces observations, il est évident qu'une solution d'automatisation intelligente s'impose pour sécuriser la saisie, accélérer le traitement et assurer la fiabilité des informations financières.

2.2.3 Solution envisagée:

Suite à la détection des contraintes du processus en cours, l'idée de solution est d'automatiser le remplissage du formulaire de saisie des factures fournisseurs. Cette automatisation serait intégrée de manière harmonieuse dans le flux actuel du module Trésorerie, section « Dépenses », sous-catégorie « Factures » du logiciel **OMRAN Estate**.

Concrètement, le principe de fonctionnement de la solution est le suivant :

- L'opérateur continue de joindre la facture comme aujourd'hui, via la fonction **Attachement**, en important le document sous différents formats ;
- Une fois le fichier importé, un module intelligent se déclenche automatiquement pour :
 - **Extraire les informations clés** : numéro et date de la facture, fournisseur, projet concerné, rubrique budgétaire, montant HT, taux de TVA, montant TTC.
 - **Détecter et vérifier les éléments visuels** : présence d'un cachet, d'une signature et du logo du fournisseur, afin de garantir l'authenticité du document.
- Toutes ces données sont ensuite structurées dans un fichier JSON généré automatiquement.
- Le JSON est alors utilisé pour remplir automatiquement le formulaire de saisie dans la rubrique Dépenses → Factures, sans intervention manuelle.

Ainsi, la solution préserve le workflow d'attachement actuel tout en y ajoutant une couche d'automatisation intelligente, afin de :

- Réduire le temps de traitement,
- Minimiser les erreurs humaines,

- Renforcer le contrôle qualité grâce à la détection automatique des éléments visuels,
- Et garantir la fiabilité et la traçabilité des données financières intégrées dans le module Trésorerie.

À terme, cette automatisation contribuera à moderniser l'utilisation d'OMRAN Estate, à améliorer la productivité des équipes comptables et à renforcer la compétitivité de « Yasmine Engineering Systems » auprès de ses clients.

En conclusion, cette analyse de l'existant a permis de mettre en évidence les limites majeures du processus actuel, justifiant ainsi le recours à une solution d'automatisation intelligente.

Dans ce qui suit, nous allons formuler de manière précise les besoins fonctionnels et non fonctionnels attendus de cette solution.

2.2.4 Définition des besoins fonctionnels et non fonctionnels

Dans le prolongement de l'analyse de l'existant et de la solution envisagée, il est essentiel de formaliser les besoins auxquels la solution doit répondre. Ceux-ci se répartissent en deux grandes catégories :

- les besoins fonctionnels, qui définissent ce que le système doit faire.
- les besoins non fonctionnels, qui précisent les exigences de qualité, de performance et de sécurité du système.

****Besoins fonctionnels****

La solution doit permettre l'automatisation intelligente du traitement des factures fournisseurs en prenant en charge les fonctions suivantes :

- **Importation des documents** : permettre à l'utilisateur d'ajouter une facture via la fonction « Attachement » du module Trésorerie, avec prise en charge des formats JPEG, PNG, etc ...
- **Déclenchement automatique du traitement** dès l'import du fichier, sans action manuelle supplémentaire.

- **Extraction automatique des informations clés** à partir de la facture, incluant :
 - Numéro de facture
 - Date d'émission
 - Nom du fournisseur
 - Montant HT
 - Taux de TVA
 - Montant TVA
 - Montant TTC
 - Projet concerné
 - Nature de la dépense
 - Référence budgétaire (si disponible)
 - Devise utilisée (DT, EUR...)
 - Conditions de paiement ou échéance
 - Code fournisseur (si présent)
- **Détection de composants visuels spécifiques** : identification du cachet, de la signature et du logo du fournisseur, afin d'assurer une vérification de l'authenticité des pièces jointes.
- **Génération d'un fichier JSON** structuré, contenant tous les champs extraits, selon un format normalisé, prêt à être intégré dans le module Trésorerie.
- **Injection automatique des données** dans le formulaire de saisie du module Trésorerie
→ Dépenses → Factures, avec validation des champs obligatoires.

****Besoins non fonctionnels****

La solution devra également répondre à des critères de performance, de sécurité et de fiabilité essentiels à son bon fonctionnement en environnement professionnel :

- **Performance** : le temps de traitement d'une facture ne devra pas excéder 10 secondes, incluant l'extraction, la structuration et l'intégration des données.
- **Précision** : le système doit atteindre un taux de fiabilité élevé dans la reconnaissance des champs, même en cas de qualité moyenne (ex. scan incliné, documents partiellement flous).
- **Robustesse** : en cas de facture incomplète, illisible ou erronée, le système devra générer une alerte sans bloquer le processus global.

- **Sécurité des données** : toutes les informations traitées doivent rester confidentielles. Aucun transfert de données vers des serveurs externes ne doit être effectué sans chiffrement ni autorisation.
- **Compatibilité** : la solution doit respecter les formats standards (PNG, JPEG etc..) utilisés dans OMRAN Estate et ne pas requérir de transformation manuelle des fichiers.
- **Intégration non intrusive** : elle doit s'intégrer dans le système actuel d'OMRAN Estate sans altérer les modules existants, ni imposer de modifications lourdes sur l'architecture logicielle.
- **Évolutivité** : la solution doit pouvoir évoluer pour intégrer d'autres types de pièces comptables à l'avenir (avoirs, bons de commande...) sans remise à plat complète du pipeline technique.
- **Ergonomie** : l'interface ou le flux d'intégration doit rester simple, intuitif et non perturbant pour les utilisateurs comptables actuels.

3 Les approches méthodologiques

Avant d'entamer notre projet, il est essentiel de définir clairement les objectifs à atteindre. Pour y parvenir, il convient de s'appuyer sur des recherches approfondies portant sur les approches méthodologiques et les bonnes pratiques en matière de science des données. Cela permet d'assurer la production d'un livrable de qualité, tout en respectant les contraintes de temps.

Il existe aujourd'hui de nombreuses méthodes adaptées aux projets de data science. Choisir la plus appropriée peut représenter un véritable défi en raison de la diversité des cadres existants.

Parmi les approches les plus couramment adoptées, on peut citer :

- **KDD**,
- **SEMMA**,
- **CRISP-DM**.

3.1 KDD : [1]

La méthodologie **KDD** (*Knowledge Discovery in Databases*) est un processus avancé visant à identifier des motifs pertinents, nouveaux, utiles et compréhensibles à partir de larges volumes

de données. Il s'agit d'une démarche itérative et interactive, allant de la compréhension du domaine jusqu'à l'interprétation et l'évaluation des résultats obtenus.

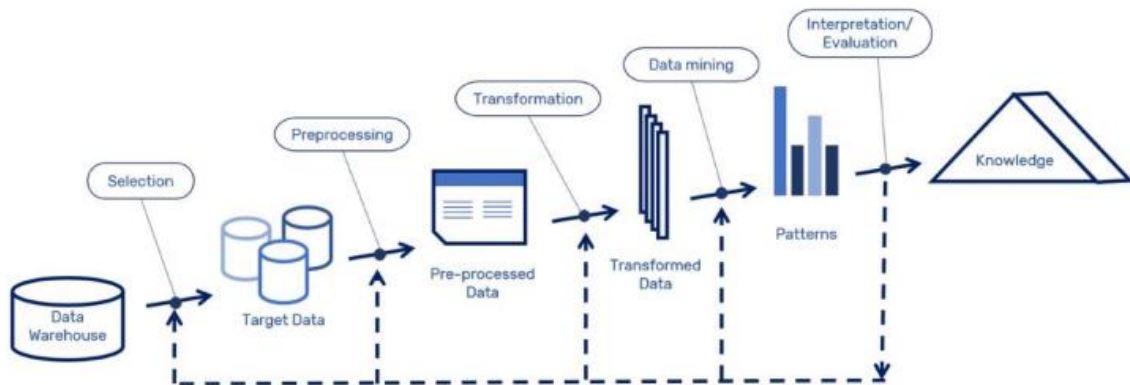


Figure 6:Processus KDD

Le processus se décompose en cinq étapes fondamentales :

1. **Sélection** : Identifier l'ensemble des données ou variables sur lesquelles l'analyse doit porter.
2. **Prétraitement** : Nettoyer et préparer les données afin d'assurer leur cohérence et leur qualité.
3. **Transformation** : Adapter les données, notamment en réduisant leur dimensionnalité, pour faciliter les analyses à venir.
4. **Exploration (Data Mining)** : Appliquer des techniques de fouille de données afin de détecter des structures ou relations pertinentes.
5. **Interprétation/Évaluation** : Analyser les motifs extraits pour juger de leur validité, nouveauté et utilité réelle.

Ce processus flexible permet aux experts de revenir à une étape antérieure si les résultats ne sont pas concluants, et d'itérer autant que nécessaire jusqu'à obtention des objectifs fixés.

3.2. SEMMA [2]

SEMMA est une méthodologie fréquemment utilisée dans l'industrie de la data science. Elle met l'accent sur l'analyse statistique et la modélisation, en se concentrant principalement sur la sélection et l'optimisation des variables influentes dans la prédiction des résultats.

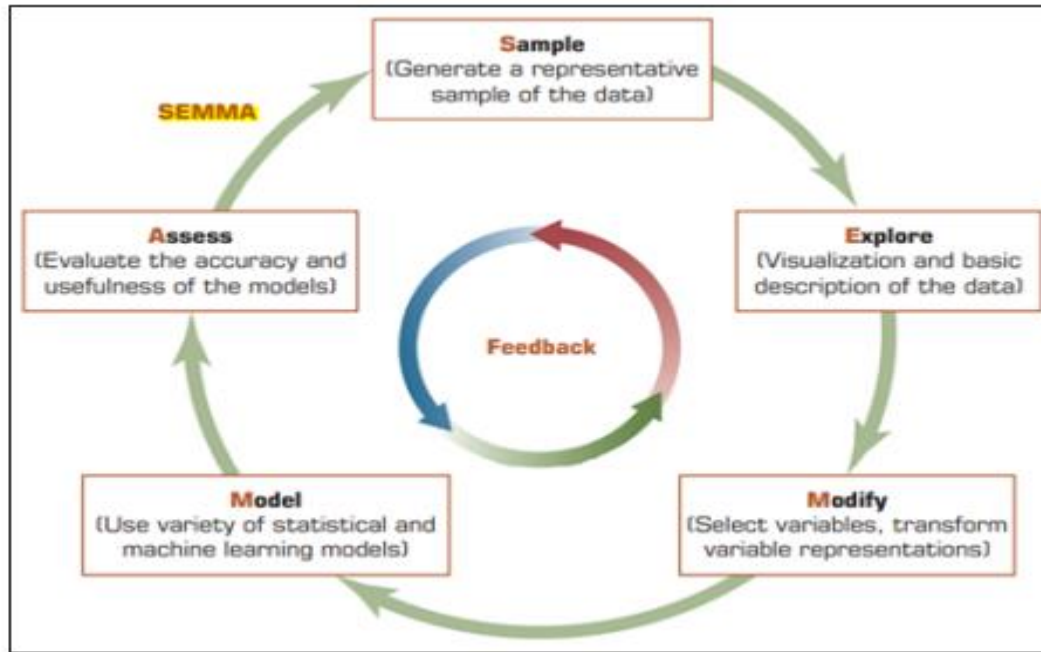


Figure 7:: Processus KDD

SEMMA est l'acronyme des cinq étapes suivantes :

1. **Sample (Échantillonnage)** : Prélever un sous-ensemble représentatif des données.
2. **Explore (Exploration)** : Étudier les données pour comprendre leur structure et identifier les variables significatives.
3. **Modify (Modification)** : Transformer les données, corriger les valeurs manquantes et générer de nouvelles variables utiles.
4. **Model (Modélisation)** : Appliquer des techniques statistiques ou de machine learning pour construire des modèles prédictifs.
5. **Assess (Évaluation)** : Tester les modèles à l'aide de jeux de validation pour mesurer leur performance.

Bien que SEMMA soit aussi un processus itératif, il est réputé pour sa simplicité et son orientation « modélisation ». Toutefois, il ne couvre pas les dimensions métier et opérationnelles, ce qui peut le rendre moins adapté à des projets à forte intégration.

3.3. CRISP-DM : [3]

CRISP-DM (*Cross Industry Standard Process for Data Mining*) est un cadre de référence largement adopté en data science. Il propose une démarche structurée, itérative et orientée

métier, qui accompagne les équipes tout au long du cycle de vie du projet — depuis la compréhension des besoins jusqu'au déploiement des résultats.

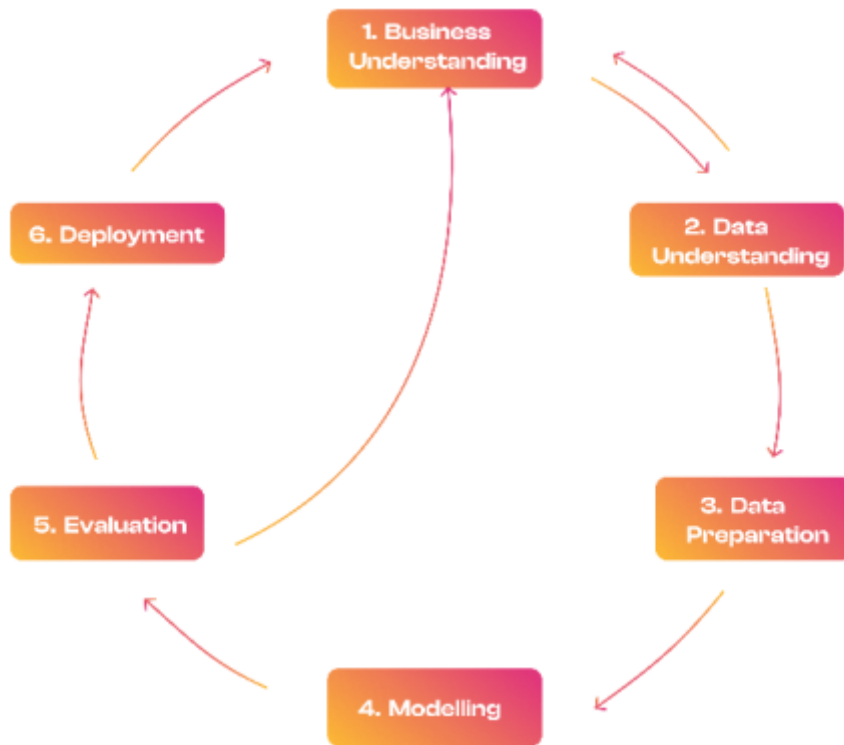


Figure 8:Processus CRISP-DM

Les six phases principales de cette méthode sont :

1. **Compréhension du besoin métier** : Collaborer avec les parties prenantes pour clarifier les objectifs, les contraintes et les résultats attendus.
2. **Compréhension des données** : Collecter, explorer et évaluer la qualité des données disponibles.
3. **Préparation des données** : Nettoyer, transformer et fusionner les sources pour construire un jeu de données exploitable.
4. **Modélisation** : Appliquer des algorithmes de machine learning adaptés au contexte, en réalisant plusieurs itérations avec la phase de préparation.
5. **Évaluation** : Vérifier si les modèles produits répondent efficacement aux objectifs définis en amont.

6. **Déploiement** : Intégrer les résultats au sein d'un système opérationnel, et produire des livrables compréhensibles pour les utilisateurs finaux.

CRISP-DM se distingue par sa rigueur, sa flexibilité et sa capacité à assurer une cohérence complète entre les exigences métiers et les traitements techniques.

3.4. Avantages et inconvénients des méthodologies

Les trois méthodologies présentent des similarités dans les étapes de traitement des données, mais se différencient par leur approche globale de la gestion de projet en science des données.

CRISP-DM se distingue par une flexibilité importante dans les transitions entre les phases, tout en intégrant des étapes dédiées à la compréhension des enjeux métiers et au déploiement de la solution. En revanche, **SEMMA** se concentre principalement sur la phase de modélisation, accordant moins d'importance aux étapes préparatoires de planification, pourtant essentielles dans l'approche CRISP-DM.

Tableau 1: Avantages et inconvénients des méthodologies

	KDD	SEMMA	CRISP-DM
Les avantages	- Processus à la fois interactif et itératif	- Il est possible de revenir à la phase d'exploration pour affiner davantage les données - Capable de gérer d'importants volumes de données - Présentation technique approfondie	- Modèle de type hiérarchique - C'est la méthode la plus couramment employée dans le processus de science des données - Très bien documentée
Les inconvénients	- Néglige certains aspects essentiels liés au domaine commercial - Les auteurs fournissent peu de détails sur les itérations entre les différentes phases	- N'inclut pas d'étape spécifique pour exploiter connaissances acquises	- Des informations restreintes concernant les boucles de retour - Le modèle doit être actualisé - Fournir une description des tâches à accomplir sans préciser les méthodes à utiliser

****Les méthodologies KDD, SEMMA et CRISP-DM structurent efficacement le cycle de vie des données, depuis la collecte jusqu'à la modélisation et l'évaluation.**

Cependant, elles se concentrent essentiellement sur l'**aspect analytique et technique** des projets de data science et **n'intègrent pas directement la gestion de l'équipe ni la planification des livraisons progressives****

Dans un projet informatique collaboratif, il est donc pertinent de s'appuyer **également sur une méthodologie de gestion de projet agile** pour :

- organiser le **travail d'équipe**,
- planifier des **incréments de développement**,
- **s'adapter rapidement** aux changements ou imprévus.

C'est dans ce contexte que Scrum trouve toute sa place.

Il complète les méthodologies analytiques **en apportant une** dimension organisationnelle et agile, **tandis que CRISP-DM reste la** méthodologie principale adoptée pour ce projet.

3.5 Méthodologie Agile Scrum: [4]

En complément des méthodologies spécifiques à la data science telles que **KDD, SEMMA et CRISP-DM**, il est nécessaire d'adopter une **méthodologie de gestion de projet** pour organiser le travail d'équipe, planifier efficacement les tâches et assurer une **livraison progressive des fonctionnalités**.

Parmi ces méthodologies, **Scrum** est l'une des plus populaires dans le **développement informatique**.

C'est un **cadre agile** reposant sur un **processus itératif et incrémental**, structuré en **Sprints**, des cycles courts de deux à quatre semaines, permettant de **livrer régulièrement des fonctionnalités opérationnelles** tout en restant **flexible face aux changements**.

Chaque sprint commence par une **planification** où l'équipe définit les objectifs à atteindre à partir du **Product Backlog**, se poursuit par des **réunions quotidiennes de suivi (Daily Scrum)** pour surveiller l'avancement et lever les blocages, et se termine par une **livraison d'un incrément fonctionnel**, suivie d'une **revue et d'une rétrospective** permettant d'améliorer le processus pour le cycle suivant.

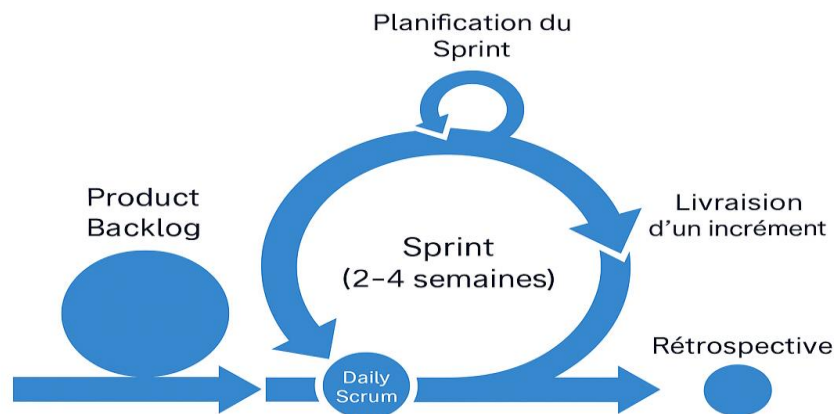
Scrum définit également trois **rôles clés** :

- **Le Product Owner**, chargé de définir les besoins et de prioriser les fonctionnalités dans le backlog.
- **Le Scrum Master**, garant du respect de la méthodologie et facilitateur pour l'équipe.
- **L'Équipe de Développement**, auto-organisée, qui prend en charge la réalisation des tâches et s'engage sur la qualité du produit.

L'adoption de Scrum permet de **réduire les risques** grâce à des livraisons incrémentales, **d'améliorer la visibilité sur l'avancement**, de **favoriser la collaboration au sein de l'équipe** et d'**augmenter la satisfaction des utilisateurs finaux**.

Dans notre projet, **Scrum constitue une approche complémentaire à CRISP-DM** : ce dernier structure le **cycle de vie des données** et les **étapes analytiques**, tandis que **Scrum** apporte une **organisation agile** pour planifier et suivre la progression des tâches.

Figure 9: Cycle Scrum illustrant les Sprints, le Product Backlog et la Rétrospective



Cycle Scrum illustrant les Sprints et la planification du projet

4 Méthodologie adoptée : CRISP-DM

Dans tout projet traitant de données volumineuses, hétérogènes et critiques pour la gestion opérationnelle, il est primordial d'adopter une méthodologie adaptée, garantissant à la fois la qualité des résultats, la maîtrise des risques et la cohérence entre les exigences métier et les solutions technologiques mises en œuvre.

Dans le cadre de notre projet d'automatisation de la saisie des factures fournisseurs au sein du module Trésorerie du logiciel OMRAN Estate, le principal défi réside dans le traitement de documents complexes et non structurés, soumis à une forte variabilité (formats multiples, présence ou non d'éléments visuels comme des cachets, logos, documents multi-pages, etc.).

Afin de relever ces défis, nous avons choisi la méthodologie **CRISP-DM (Cross-Industry Standard Process for Data Mining)**, largement reconnue pour sa robustesse dans la conduite de projets liés à l'intelligence artificielle, à la vision par ordinateur et à l'apprentissage automatique.

Raisons du choix de CRISP-DM

- **Gestion de la complexité documentaire**

- Le traitement automatisé des factures dans le secteur immobilier implique une diversité de formats, de structures et de contenus. CRISP-DM permet de segmenter le projet en étapes claires et maîtrisables.

- **Assurance qualité des données**

- Elle garantit une préparation rigoureuse des données, indispensable pour des extractions fiables.

- **Exploration intelligente des documents**

- L'analyse exploratoire permet d'identifier les variations de structure ou d'éléments visuels entre documents.

- **Modélisation multicouches**

- La méthodologie permet d'intégrer plusieurs approches (OCR, détection d'objets, LLM) dans un pipeline unique et cohérent.

- **Évaluation rigoureuse**

- Elle permet de valider la précision des extractions et la robustesse du système en conditions réelles.

- **Déploiement fluide**

- La transition vers l'environnement de production est facilitée, sans rupture dans les processus actuels.

4.1 Mise en œuvre concrète

Le présent projet repose sur la méthodologie CRISP-DM (Cross-Industry Standard Process for Data Mining), largement reconnue dans le domaine de la science des données et des projets IA. Cette approche structurée et itérative a été choisie pour assurer une cohérence claire entre les

objectifs métier et le pipeline technique, de la compréhension du besoin à l'intégration réelle dans le module Trésorerie du logiciel OMRAN Estate.

4.1.1 Compréhension du problème métier

Cette phase consiste à identifier précisément les besoins réels et à bien comprendre le processus métier actuel.

Dans notre cas :

- Étudier comment les factures sont traitées manuellement,
- Identifier les points bloquants (saisie chronophage, erreurs, doublons),
- Définir les résultats attendus (remplissage automatique du formulaire, génération JSON).

4.1.2 Compréhension des données

L'objectif est d'explorer et d'évaluer la qualité des données disponibles :

- Recenser les différents types de documents à traiter (images scannées, formats mixtes);
- Analyser la structure et le contenu des factures ;
- Détecter d'éventuelles incohérences ou doublons ;

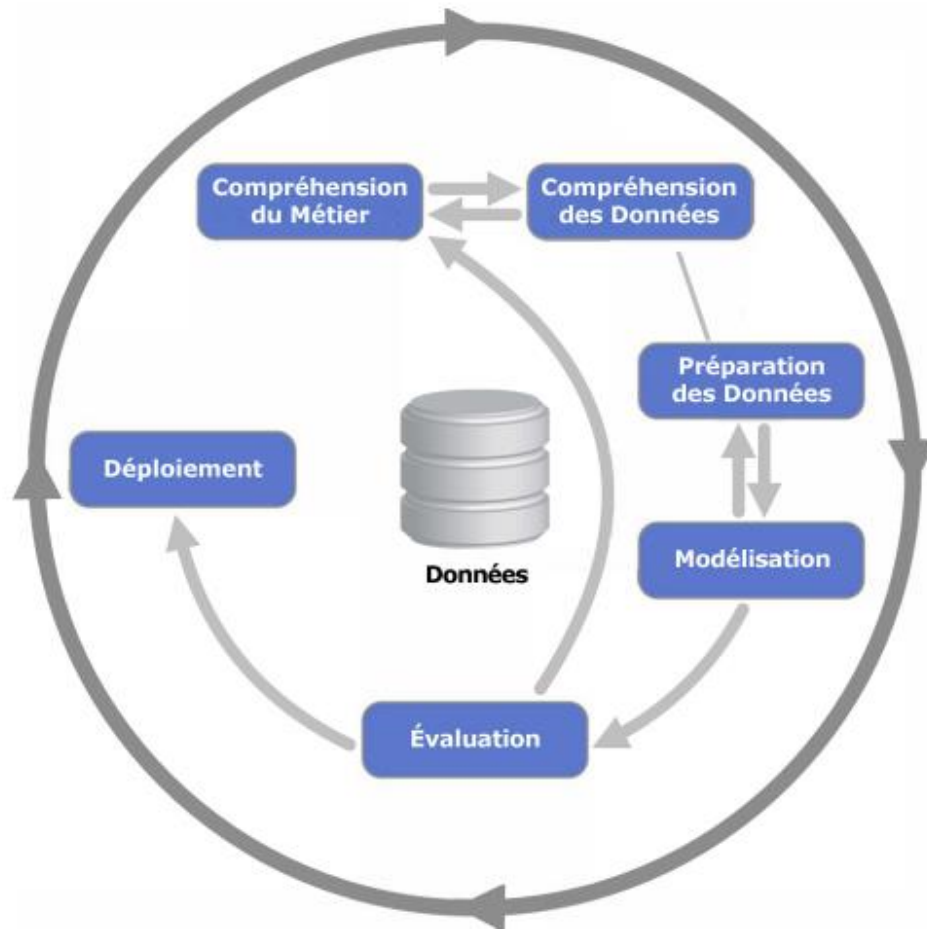


Figure 10: Les différentes phases de la méthodologie CRISP-DM

4.1.3 Préparation des données

Cette étape est clé pour la réussite technique :

- Annotation manuelle pour classifier les types de documents,
- Nettoyage et normalisation des formats,
- Classification des documents

4.1.4 Modélisation

Durant cette phase, plusieurs modèles ont été mis en œuvre :

- **OCR avancé** (*PaddleOCR*) pour le texte,
- **Détecteurs d'objets** (*YOLO v5/v8*) pour logos, tampons et signatures,
- **Modèles LLM** (*Gemma, Gemini*) pour l'interprétation et la vérification des données clés.

=>Tous ces modèles ont été testés et optimisés pour garantir un haut niveau de précision.

4.1.5 Évaluation

L'évaluation mesure la performance réelle du pipeline :

- Précision de détection d'objets visuels,
- Fiabilité des champs extraits,

4.1.6 Déploiement

La solution développée prend la forme d'une **application web légère**, composée :

- d'une interface utilisateur en Next.js, qui sert uniquement à simplifier le pipeline et à mieux visualiser le fonctionnement réel du traitement ;
- d'un backend en FastAPI, qui assure l'extraction des informations clés, la génération d'un fichier JSON structuré et l'intégration des données dans le module Trésorerie du logiciel OMRAN Estate.

L'interface permet d'enchaîner visuellement les principales étapes du processus :

- **classification automatique du type de document** à partir du texte détecté (via PaddleOCR),
- **affichage des objets détectés** (cachet, logo, signature) via YOLOv8,
- **présentation des informations extraites directement sous format JSON.**

4.2 Conclusion sur le choix

En résumé, CRISP-DM a permis de cadrer rigoureusement toutes les étapes du projet, tout en intégrant des outils de Machine Learning, de Computer Vision et de LLM pour automatiser la saisie et l'exploitation des factures.

Cette approche garantit la qualité, la traçabilité et l'adéquation de la solution aux exigences concrètes de « Yasmine Engineering Systems »

5 Environnement logiciel et Technique:

Cette partie décrit le cadre technologique utilisé pour l'exécution du projet. Elle fournit une description des langages, outils, bibliothèques et de l'infrastructure matérielle nécessaires pour la conception, l'entraînement et le déploiement du processus d'extraction automatique des données à partir d'images.

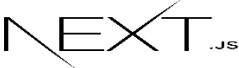
Ce cadre a été sélectionné afin d'assurer la performance du processus, la simplification du développement et l'incorporation aisée de la solution au module Trésorerie du logiciel en place.

5.1 Environnement logiciel

5.1.1 Logiciels utilisés

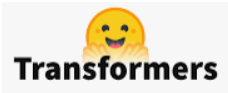
Tableau 2: les logiciels utilisés.

Logiciel	Description
Visual Studio Code 	Éditeur de code source développé par Microsoft, multiplateforme et extensible, avec des fonctionnalités comme le débogage, Git, et l'IntelliSense.
Jupyter Notebook 	Application web interactive permettant de créer et partager du code, des visualisations et du texte, compatible avec Python, Julia, R, etc.
Labellmg 	Outil open source d'annotation d'images permettant de dessiner des bounding boxes et de générer des fichiers au format YOLO ou Pascal VOC.
Docker Desktop 	Outil local permettant d'exécuter des conteneurs Docker sur un environnement Windows, avec intégration WSL2.
Microsoft Word 	Logiciel de traitement de texte de la suite Microsoft Office, utilisé pour la rédaction de documents techniques.

Next.js		Framework React.js permettant de développer des applications web performantes avec rendu côté serveur, routing simplifié et génération statique.
---------	---	--

5.1.2 Bibliothèques utilisées

Tableau 3: Bibliothèques utilisées

Bibliothèque	Description
Ultralytics	Framework open source pour la détection d'objets basé sur YOLOv8.
	
PaddleOCR	Système OCR open source offrant détection de texte, classification et reconnaissance.
	
Pytesseract	Wrapper Python pour Tesseract OCR développé par Google.
	
Transformers	Bibliothèque de modèles NLP préentraînés (Qwen, Gemini, BERT, GPT, etc.).
	
huggingface_hub	Interface pour télécharger des modèles depuis Hugging Face.
	
opencv-python / Pillow	Bibliothèques de traitement d'image.
	
FastAPI / Uvicorn	Framework web moderne et serveur ASGI pour exposer une API rapide en Python.
	
matplotlib / seaborn	Outils de visualisation de données.
	

numpy / pandas	 	Librairies pour le calcul scientifique et la manipulation de données tabulaires.
torch / torchvision		Framework de deep learning et outils pour le traitement d'image.

5.1.3 Langages utilisés

Tableau 4: Langages utilisés

Langage	Description
	Langage principal utilisé pour l'implémentation des modèles OCR, de détection d'objets, de traitement des données, ainsi que pour le backend via FastAPI.
 	Utilisé pour le développement de l'interface web avec React via le framework Next.js. JSX permet d'intégrer du HTML directement dans le code JavaScript.
 	Langages de structuration et de mise en forme utilisés pour construire et styliser l'interface utilisateur.
	Langage de balisage utilisé pour la documentation technique (README, cahiers de tests, commentaires)
Dockerfile 	Fichier de script permettant de définir une image Docker et d'automatiser la construction du conteneur du backend
	Format d'échange de données structuré, utilisé pour représenter et transmettre les informations extraites automatiquement des

5.2. Équipement matériel

Le développement, les tests et les expérimentations ont été réalisés à l'aide d'une station de travail locale et de ressources cloud. La machine principale utilisée possédait les caractéristiques suivantes :

- Processeur : **Intel Core i7 (10e génération)**
- Mémoire vive (RAM) : **16 Go**
- Disque SSD : **512 Go**
- Carte graphique (GPU) : **NVIDIA GTX 1650**
- Système d'exploitation : Windows 11

Conclusion

Dans ce premier chapitre, nous avons posé les bases de notre projet en présentant l'organisme d'accueil, **Yasmine Engineering Systems**, ainsi que le **contexte fonctionnel et technologique** du besoin métier identifié. À travers une étude approfondie de l'existant, nous avons mis en évidence les limites du processus manuel de traitement des factures fournisseurs, ce qui a motivé la proposition d'une solution d'**automatisation intelligente**.

Nous avons ensuite formalisé les **besoins fonctionnels et non fonctionnels** de cette solution, avant de justifier le choix de la **méthodologie CRISP-DM** pour structurer notre démarche. Enfin, nous avons présenté l'**environnement logiciel et technique** dans lequel le projet sera développé, assurant ainsi la faisabilité de l'implémentation.

Le chapitre suivant se penchera sur les **notions théoriques essentielles** à la compréhension des technologies utilisées, notamment l'OCR, la vision par ordinateur et le traitement du langage naturel.

Chapitre 2

Etude de l'état de l'art et fondements théoriques

Introduction

Ce chapitre présente les concepts fondamentaux sur lesquels repose notre projet, à savoir l'intelligence artificielle et l'apprentissage profond, appliqués à la vision par ordinateur et au traitement automatique du langage. Nous y détaillons les principales familles de modèles utilisées pour la **détection d'objets visuels** (logos, signatures, cachets) ainsi que les approches modernes permettant l'**extraction d'informations** à partir de documents numérisés. Ces fondements théoriques constituent la base du système développé dans les chapitres suivants.

1 Introduction à l'intelligence artificielle

1.1. Définition de l'intelligence artificielle et ses sous-domaines

L'intelligence artificielle (IA) désigne un ensemble de techniques visant à reproduire des capacités cognitives humaines telles que la perception, l'apprentissage, le raisonnement ou la prise de décision à travers des systèmes informatiques. Elle repose sur des algorithmes capables d'analyser des données, de détecter des régularités et de prendre des décisions basées sur l'expérience.

L'IA se divise en plusieurs sous-domaines, dont les plus connus sont :

- Le **machine learning (ML)**, ou apprentissage automatique, où les modèles apprennent à partir de données plutôt que de règles prédéfinies ;
- Le **traitement automatique du langage naturel (NLP)**, qui s'intéresse à l'interprétation des textes et de la parole ;
- La **vision par ordinateur (Computer Vision)**, qui permet à la machine d'interpréter des images et des vidéos.

1.2. Les types d'apprentissage en IA

Trois grands paradigmes d'apprentissage sont utilisés en IA :

- **Apprentissage supervisé** : les données d'entraînement sont étiquetées, ce qui permet au modèle d'apprendre à prédire une sortie à partir d'une entrée.
- **Apprentissage non supervisé** : aucune étiquette n'est fournie ; le modèle tente de découvrir des structures cachées dans les données.

- **Apprentissage par renforcement** : l'agent apprend en interagissant avec un environnement et en recevant des récompenses ou pénalités.

1.3. L'émergence du Deep Learning

Le *Deep Learning*, ou *apprentissage profond*, est une branche du *Machine Learning* qui repose sur l'utilisation de réseaux de neurones artificiels comportant plusieurs couches. Grâce à leur architecture hiérarchique, ces modèles peuvent apprendre des représentations de plus en plus abstraites à partir de données brutes.

Cette approche a permis des avancées spectaculaires dans des domaines complexes tels que la reconnaissance d'images, la traduction automatique ou encore la compréhension du langage naturel. Elle constitue aujourd'hui le fondement technologique de nombreux systèmes intelligents modernes.

La figure n10 ci-après illustre cette hiérarchie entre l'intelligence artificielle, le machine learning et le deep learning.

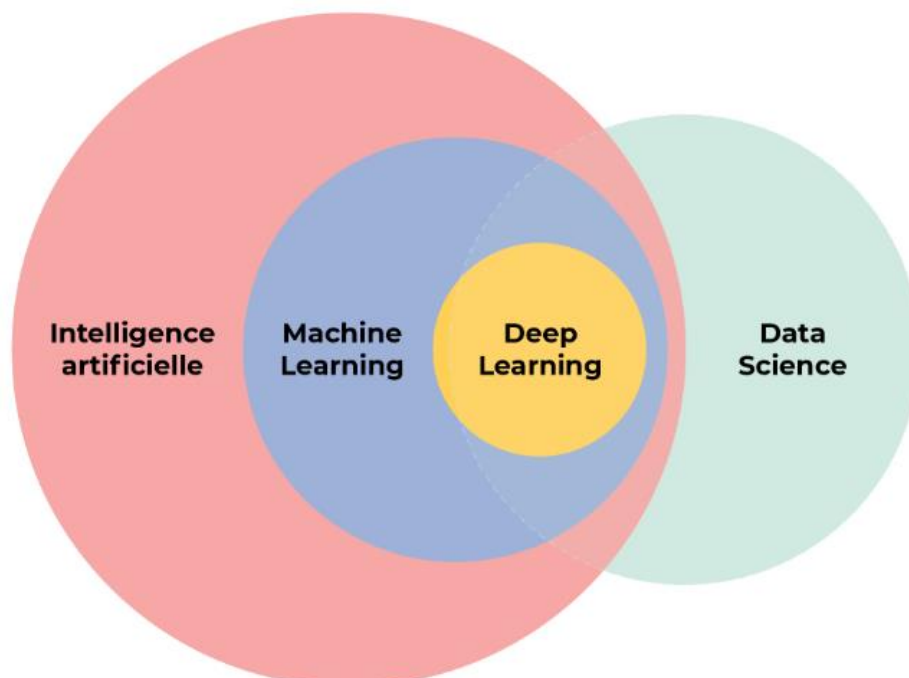


Figure 11: Hiérarchie conceptuelle de l'intelligence artificielle et de ses sous-domaines

2 Familles de modèles d'apprentissage profond

Cette section présente les grandes familles d'architectures utilisées dans les projets d'IA modernes, en particulier pour le traitement de l'image et du texte.

2.1. Réseaux de neurones artificiels (ANN) [5]

Les *Artificial Neural Networks* (ANN) constituent l'architecture de base de l'apprentissage profond. Ils sont composés de couches de neurones artificiels connectés entre eux, où chaque neurone effectue une transformation linéaire suivie d'une fonction d'activation non linéaire. Lorsqu'ils comportent de nombreuses couches cachées, on parle alors de *réseaux profonds* (*deep neural networks*). Bien que puissants pour des tâches simples, les ANN standards ne sont pas optimisés pour des données structurées comme les images ou les séquences temporelles, d'où la nécessité d'architectures spécialisées.

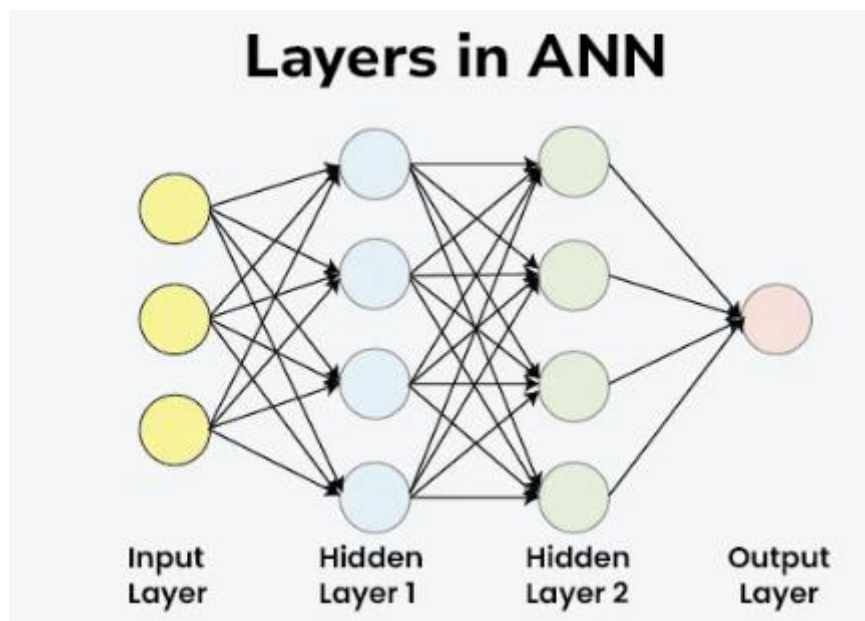


Figure 12: Architecture typique d'un réseau de neurones artificiel (ANN)

Légende :

Cette figure représente un réseau de neurones artificiel standard. On y distingue la couche d'entrée, une ou plusieurs couches cachées, et la couche de sortie. Chaque neurone est connecté à tous les neurones de la couche suivante, illustrant la propagation de l'information dans un réseau dense.

2.2 Réseaux convolutifs (CNN) [6]

Les *Convolutional Neural Networks* (CNN) sont des architectures spécialisées dans le traitement des données visuelles. Grâce à l'application de filtres convolutifs glissants, les CNN capturent les motifs spatiaux locaux dans les images (bords, textures, formes), tout en réduisant le nombre de paramètres grâce au partage des poids. Les couches de *pooling* permettent une réduction de la dimensionnalité, ce qui limite le surapprentissage et améliore la robustesse du modèle. Enfin, les couches entièrement connectées situées en sortie sont responsables de la prise de décision, typiquement pour des tâches de classification.

La figure 2.3 ci-après illustre l'architecture typique d'un CNN utilisé pour la reconnaissance d'images.

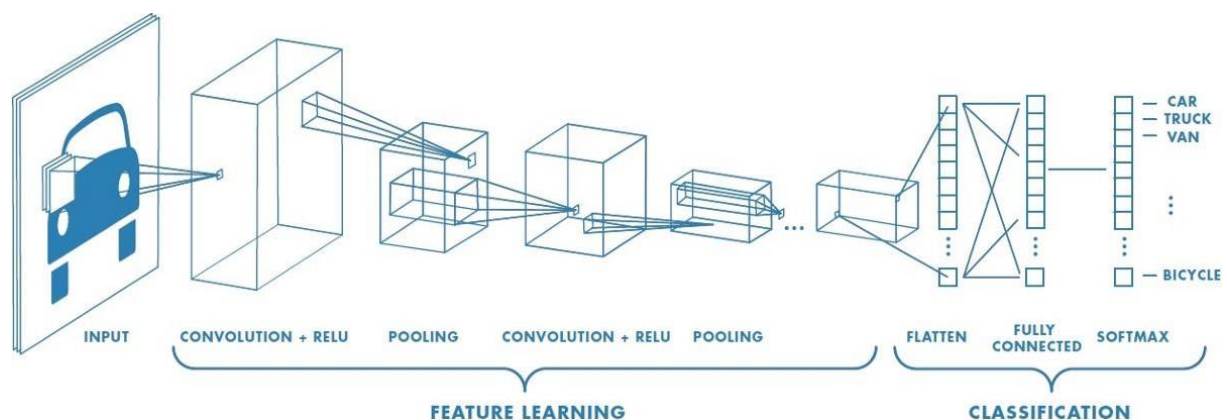


Figure 13: CNN Architecture

2.3. Réseaux récurrents (RNN) et variantes [7]

Les **Recurrent Neural Networks** (RNN) sont conçus pour traiter des séquences temporelles ou textuelles. Contrairement aux réseaux classiques, les RNN possèdent une mémoire interne qui leur permet de conserver le contexte des entrées précédentes. Cependant, ils sont sujets à

des problèmes d'instabilité numérique comme la disparition ou l'explosion du gradient sur les longues séquences.

Pour pallier ces limitations, des variantes plus stables et performantes ont été développées, notamment : [7]

- **LSTM (Long Short-Term Memory)** : intègre des mécanismes de portes (input, forget, output) pour mieux gérer les dépendances à long terme.
- **GRU (Gated Recurrent Unit)** : simplifie la structure des LSTM tout en conservant des performances comparables.

Bien que les RNN et leurs variantes soient efficaces, ils sont progressivement supplantés par les **Transformers**, plus adaptés au traitement parallèle et aux longues dépendances contextuelles.

2.4. Transformers et modèles de langage

Les **Transformers** [8], introduits par Vaswani et al. en 2017 dans l'article fondateur "*Attention is All You Need*", ont marqué une révolution dans le traitement des données séquentielles, en particulier du langage naturel. Contrairement aux architectures RNN, les Transformers s'affranchissent du traitement séquentiel grâce à un mécanisme d'**attention** qui permet à chaque élément d'une séquence de se connecter directement à tous les autres, quels que soient leur position ou leur éloignement.

L'architecture Transformer repose principalement sur deux blocs :

- **Self-Attention** : pondère chaque token de la séquence en fonction de sa relation contextuelle avec les autres ;
- **Feed-forward network** : couche dense indépendante pour chaque position, permettant l'apprentissage de représentations riches.

Grâce à leur structure entièrement parallélisable, les Transformers permettent des entraînements massifs sur des corpus gigantesques, avec une efficacité inégalée. Ils sont à la base des **modèles de langage de grande taille (LLM)** tels que :

- **BERT** [9] (modèle encodeur bidirectionnel pour la classification et l'extraction),

- **GPT** (modèle auto-régressif pour la génération de texte),
- **T5** (modèle unifié texte-à-texte),
- **Gemini** (multimodal, orienté usages applicatifs).

La figure n14 ci-dessous illustre les différences structurelles entre le Transformer standard, BERT et GPT.

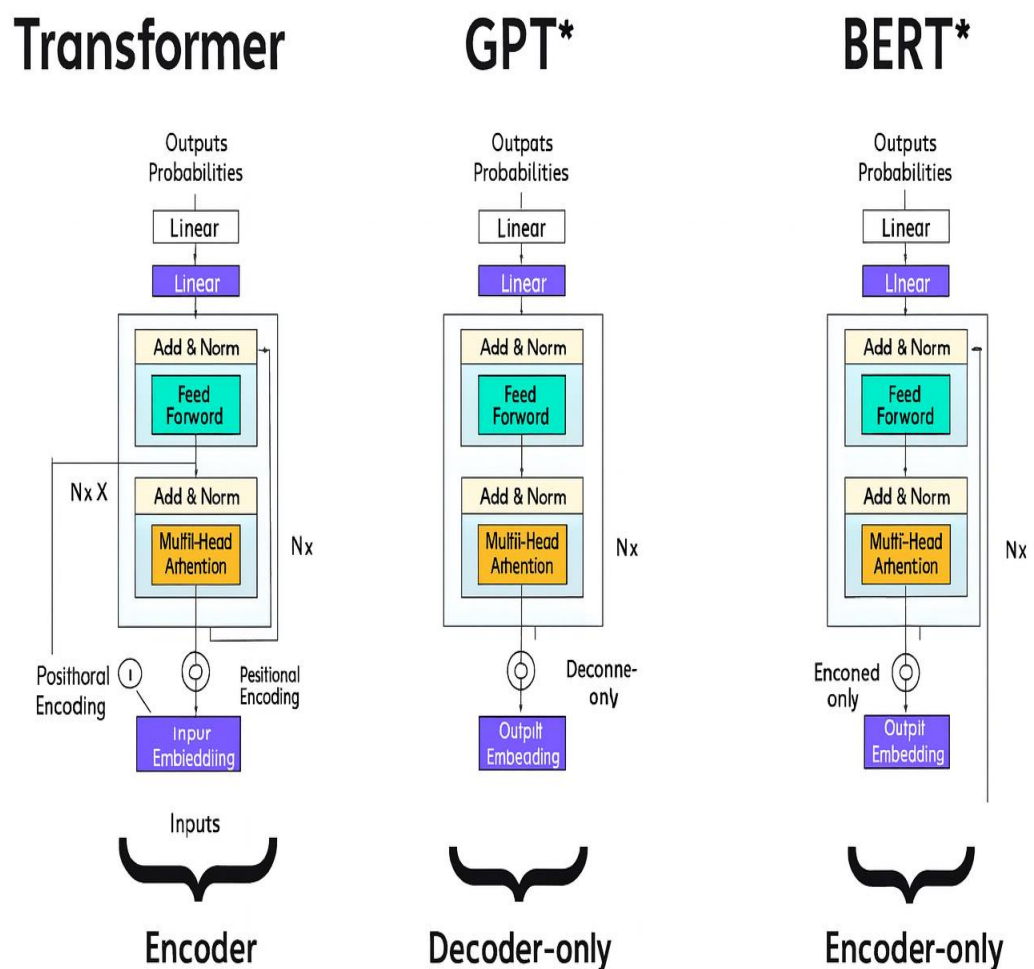


Figure 14: Architectures du Transformer, de GPT et de BERT

“Cette figure compare trois variantes de l’architecture Transformer. Le modèle original (à gauche) associe un encodeur et un décodeur, adapté aux tâches de traduction. GPT (au centre) utilise uniquement le décodeur pour générer du texte de manière autorégressive. BERT (à

droite) repose exclusivement sur l'encodeur pour capturer le contexte bidirectionnel des séquences, ce qui le rend efficace pour les tâches de compréhension du langage.”

Aujourd'hui, les Transformers constituent le socle technologique central de la majorité des applications intelligentes dans les domaines du langage, de la vision et de la reconnaissance de documents.

Cette architecture puissante est à l'origine de nombreuses avancées récentes dans le **traitement automatique du langage naturel (NLP)**, domaine dans lequel les Transformers ont permis l'émergence des **modèles de langage de grande taille (LLM)**. La section suivante détaille ces applications et les modèles associés.

2.5. Traitement du langage naturel (NLP) et LLM

Le traitement du langage naturel repose aujourd'hui principalement sur les modèles basés sur les Transformers, qui ont permis l'émergence des LLM. Cette section présente les concepts fondamentaux du NLP, les tâches classiques, puis les modèles de langage de grande taille et leurs applications concrètes.

2.5.1. Traitement automatique du langage naturel (NLP) [10]

Le **traitement automatique du langage naturel (Natural Language Processing – NLP)** est une branche de l'intelligence artificielle qui permet aux ordinateurs de comprendre, interpréter, manipuler et générer du langage humain. Il repose sur des méthodes issues de la linguistique, de la statistique et de l'apprentissage automatique, et il est essentiel pour de nombreuses tâches telles que :

- L'analyse syntaxique et sémantique
- La classification de texte
- L'extraction d'informations
- La traduction automatique
- La génération de résumés ou de réponses automatiques

Le NLP est à la base des systèmes de dialogue, des moteurs de recherche intelligents, ou encore des assistants vocaux. Grâce aux progrès du deep learning, et notamment à l'introduction des Transformers, ses capacités se sont considérablement accrues.

2.5.2. Tâches classiques du NLP

- **NER (Named Entity Recognition)** : détection et classification automatique des entités nommées (noms de personnes, dates, montants, lieux, etc.) dans un texte. Cruciale pour l'extraction d'informations dans les documents administratifs.
- **Extraction relationnelle** : identification des relations entre entités extraites, par exemple la relation fournisseur–montant dans une facture.
- **Résumé automatique** : génération d'un condensé des informations clés à partir d'un long document, en préservant le sens général.
- **Classification thématique** : assignation automatique d'un texte à une ou plusieurs catégories prédéfinies (ex. : facture, devis, contrat).
- **Traduction automatique** : conversion d'un texte d'une langue source vers une langue cible à l'aide de modèles neuronaux séquence-à-séquence.
- **Question-réponse (Q&A)** : systèmes capables de répondre à des questions posées en langage naturel en s'appuyant sur un contexte textuel donné.

2.5.3. Modèles de langage de grande taille (LLM) [11]

Les **modèles de langage de grande taille (LLM – Large Language Models)** sont des réseaux neuronaux entraînés sur des corpus textuels massifs, capables de comprendre et de générer du langage naturel avec une cohérence contextuelle élevée. Reposant sur l'architecture Transformer, ils ont révolutionné le traitement du langage à grande échelle.

Les LLM peuvent être utilisés de deux manières principales :

- Via des **API commerciales** (ex. : GPT d'OpenAI, Claude d'Anthropic, Gemini de Google), accessibles en ligne avec authentification et tarification à l'usage,
- Via des **modèles open-source exécutés localement ou dans le cloud**, grâce aux bibliothèques comme **Hugging Face Transformers**, qui permettent de charger, configurer, et déployer des LLM (Gemma, Mistral, Falcon, etc.) dans des environnements maîtrisés.

Ces modèles sont utilisés dans des cas variés : génération de contenu, traduction, recherche sémantique, complétion de texte, extraction d'informations ou encore génération de code.

Les LLM sont utilisés dans des cas variés : génération de contenu, traduction, recherche sémantique, complétion de texte, extraction d'informations ou encore génération de code.

Les modèles actuels les plus représentatifs incluent :

- **GPT (OpenAI)** : une série de modèles de langage auto-régressifs développés par OpenAI, à commencer par GPT-1 jusqu'à GPT-4, conçus pour générer du texte cohérent, compléter des phrases, répondre à des questions ou encore rédiger automatiquement des documents. GPT est basé sur l'architecture Transformer et s'illustre par sa capacité à modéliser le langage de manière contextuelle grâce à un entraînement massif non supervisé sur de vastes corpus textuels.
- **Claude (Anthropic)** : une famille de modèles conversationnels développée par Anthropic, basée sur une approche centrée sur la sécurité, la constitutionnalité et la transparence. Claude a été conçu comme une alternative éthique à GPT, avec une attention particulière à la robustesse aux dérives et une meilleure explicabilité des réponses. Son nom rend hommage à Claude Shannon, pionnier de la théorie de l'information.
- **Gemini (Google DeepMind)** : modèle multimodal de nouvelle génération développé par Google, capable de comprendre et de générer du texte, du code, des images et des données audio. Introduit comme successeur de Bard, Gemini combine la puissance des Transformers avec des mécanismes d'apprentissage efficaces pour intégrer plusieurs types de données, facilitant les applications documentaires, conversationnelles et analytiques.
- **Gemma (Google, open-source)** : série de modèles de langage open-source proposée par Google pour des usages locaux, embarqués ou souverains. Gemma 2B et 7B sont optimisés pour un déploiement sur machines personnelles ou serveurs, et visent à démocratiser l'accès aux LLM performants tout en assurant la confidentialité des données. Ils sont compatibles avec les bibliothèques Hugging Face et permettent une fine personnalisation.

Ils permettent de créer des agents conversationnels puissants, des assistants métier ou encore des systèmes de traitement automatique de documents.

- **Extraction relationnelle**
- **Résumé automatique**

- **Classification thématique**

- ** Focus sur Gemini**

Gemini est une série de modèles multimodaux de Google DeepMind, capables de traiter texte, image, code, audio.

- **Gemini 1.5 Pro** : modèle avancé, comparable à GPT-4 Turbo
- **Gemini Flash** : version légère pour traitement rapide

Exemple de coût d'appel API (juillet 2025) :

- 1000 tokens en entrée : 0,0005 \$
- 1000 tokens en sortie : 0,0015 \$ => Une facture (700 in / 400 out) à environ **0,001 \$**

- ** Focus sur Gemma**

Gemma est une famille de modèles de langage de grande taille open-source développée par Google DeepMind. Conçus pour être légers, efficaces et facilement intégrables dans des pipelines IA personnalisés, les modèles Gemma sont adaptés à l'exécution locale, ce qui les rend idéaux pour les entreprises recherchant une maîtrise totale de leurs données et coûts.

Deux principales variantes sont disponibles :

- **Gemma 3B** : optimisé pour tourner sur des CPU et petits GPU, pratique pour les tests ou les déploiements à faible coût.
- **Gemma 7B** : plus grand et plus performant, destiné à un usage sur des serveurs ou infrastructures cloud plus puissantes.

Gemma se distingue par sa licence permissive, sa compatibilité avec les bibliothèques Hugging Face Transformers et son intégration aisée dans des applications IA hybrides (OCR + NLP).

Avantages : gratuit, exécutable localement, adaptable.

2.6. Vision par ordinateur (Computer Vision) [12]

La **vision par ordinateur (Computer Vision)** est un domaine de l'intelligence artificielle qui vise à permettre aux machines d'interpréter et de comprendre le contenu d'images ou de vidéos. Elle repose principalement sur des algorithmes de traitement d'image, de reconnaissance de motifs, et plus récemment sur des réseaux neuronaux profonds comme les **CNN** et **Transformers visuels (ViT)**.

2.6.1. Objectifs et enjeux

La vision par ordinateur vise à extraire des informations utiles à partir de contenus visuels afin de permettre à un système de prendre des décisions ou de produire une sortie structurée. Ses applications sont nombreuses :

- Reconnaissance faciale et biométrie
- Classification et détection d'objets
- Suivi de mouvement
- Inspection industrielle
- Lecture de texte (OCR)
- Analyse de scènes

2.6.2 Object Detection

La **détection d'objets** constitue une tâche essentielle en vision par ordinateur, visant à identifier et localiser automatiquement des entités visuelles spécifiques au sein d'une image. Elle joue un rôle clé dans le traitement de documents numérisés, notamment pour l'identification de zones graphiques comme les logos, cachets ou signatures.

Les modèles de détection d'objets les plus performants s'appuient sur des architectures de réseaux neuronaux convolutifs (CNN)

Parmi les architectures les plus répandues figurent YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), Faster R-CNN (Region-based Convolutional Neural Network), etc...

Ces modèles, généralement pré-entraînés sur de grands corpus d'images annotées, sont capables de localiser une diversité d'objets avec un haut niveau de précision, et peuvent être réentraînés ou adaptés à des cas d'usage spécifiques.

Une comparaison entre trois modèles de référence basés sur les CNN — YOLO, SSD et Faster R-CNN — permet de mettre en évidence les compromis existants entre précision, vitesse d'exécution et complexité algorithmique.

Parmi eux,

- ✓ **Faster R-CNN** [13] est un modèle fondamental dans le domaine de la détection d'objets proposé par Shaoqing Ren et al. en 2015. Il marque une avancée majeure en introduisant une méthode **intégrée et efficace** pour générer des régions d'intérêt (regions of interest) de manière automatique, tout en conservant une **précision élevée**, notamment sur des objets de petite taille ou dans des environnements complexes.

Principe général

Faster R-CNN repose sur une approche **two-stage** (en deux étapes) :

1. **Le RPN (Region Proposal Network)** génère des propositions de régions où des objets pourraient se trouver.
2. Ces propositions sont ensuite affinées et classées par un **réseau de détection**, qui prédit la classe de l'objet et ajuste les coordonnées de la boîte englobante.

Cette séparation en deux phases permet au modèle de se concentrer sur les régions les plus prometteuses, améliorant ainsi la qualité des prédictions.

Composants de l'architecture

L'architecture de Faster R-CNN est composée des éléments suivants :

1. Couches convolutives partagées

Un réseau de neurones convolutif (souvent **VGG** ou **ResNet**) extrait des **caractéristiques visuelles** de l'image. Ces features sont utilisées à la fois par le RPN et le réseau de détection, ce qui rend le processus **plus rapide et cohérent**.

2. RPN (Region Proposal Network)

Le RPN analyse la carte de caractéristiques pour générer des **propositions de régions** (bounding boxes) à partir d'un ensemble de **boîtes d'ancrage** (anchors) de tailles et proportions variées. Il évalue chaque ancre selon :

- sa probabilité de contenir un objet,
- les ajustements nécessaires à sa forme.

3. RoI Pooling

Cette couche transforme chaque région proposée par le RPN en une **zone de taille fixe**, adaptée aux couches entièrement connectées. Cela permet un traitement uniforme des régions, quelle que soit leur taille initiale.

4. Réseau de détection final

Chaque **Region of Interest (RoI)** est transmise à un classifieur qui :

- **prévoit la classe** de l'objet (ou "fond" s'il n'y en a pas),
- **ajuste précisément** les coordonnées de la boîte de détection (régression de boîte).

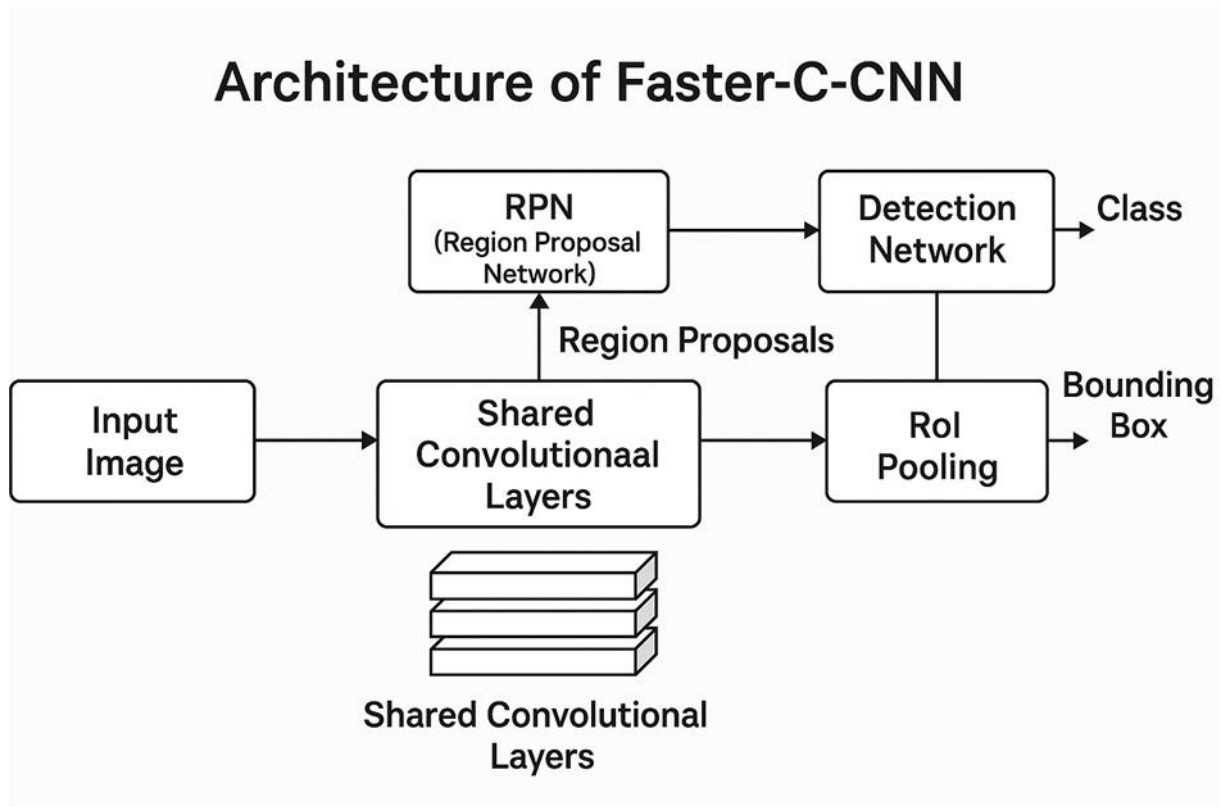


Figure 15: architecture Faster R-CNN

Avantages de Faster R-CNN

-  **Très haute précision**, surtout sur des objets petits, partiellement visibles ou dans des scènes denses.
-  **Architecture modulaire** : on peut facilement changer le backbone (ResNet, MobileNet...), adapter le nombre de classes, ou affiner l'entraînement.
-  Bonne **cohérence spatiale** grâce au mécanisme de RoI et au traitement ciblé des régions.

Limites de Faster R-CNN

- **Temps d'inférence élevé** (~5–7 FPS sur une NVIDIA GTX 1650), ce qui le rend inadapté pour les applications **temps réel** (vidéosurveillance en direct, robotique, etc.).
- Moins performant que les modèles “one-stage” (comme SSD ou YOLO) en termes de vitesse et de simplicité de déploiement.

- **Entraînement plus complexe** : plus de composants, plus d'étapes à maîtriser (anchors, RoI, loss multiples...).

Conclusion

Faster R-CNN reste un **modèle de référence** en matière de détection d'objets grâce à sa **précision exceptionnelle**. Il est particulièrement adapté aux tâches qui exigent une **localisation fine et fiable**, comme **l'analyse d'imagerie médicale, la reconnaissance de visages ou la détection dans des images complexes**.

Cependant, pour un usage plus dynamique, comme la détection **en temps réel** sur flux vidéos ou des images ou déploiement sur machines à ressources limitées, d'autres alternatives comme **SSD** ou **YOLO** sont plus adaptées.

- ✓ SSD (Single Shot MultiBox Detector) [14]

Le **SSD** (*Single Shot MultiBox Detector*) est un algorithme de détection d'objets introduit en 2016 par **Wei Liu et al.**, basé sur les réseaux neuronaux convolutifs (CNN). Il se distingue des modèles plus complexes comme Faster R-CNN par son fonctionnement en **une seule étape** (*single shot*), ce qui lui confère une **grande rapidité d'exécution**, sans compromettre la précision de manière significative.

Fonctionnement général

Contrairement aux méthodes à deux phases (propositions puis classification), SSD exécute **simultanément** deux tâches sur une même image :

1. **Localiser** les objets à l'aide de **boîtes englobantes** (bounding boxes) prédéfinies ;
2. **Classer** ces objets parmi les différentes catégories possibles (ex. : signature, logo, cachet, etc.).

Pour cela, SSD applique des **convolutions** sur plusieurs couches du réseau, chacune étant responsable de détecter des objets à différentes **échelles**. Les couches proches de l'entrée détectent des objets **grands**, tandis que les couches plus profondes ciblent les **petits objets**. Cela rend le modèle robuste à la **variabilité de taille**.

Architecture technique

L'architecture SSD est composée de deux blocs principaux :

- Une **backbone CNN** (ex. VGG16 ou MobileNet) qui extrait les caractéristiques visuelles de l'image.
- Un ensemble de **couches supplémentaires** qui génèrent des prédictions de localisation (coordonnées des boîtes) et de classification (probabilités de classes), sur **plusieurs niveaux de la pyramide de caractéristiques**.

Chaque position de la grille génère plusieurs "**default boxes**" (ancres) à divers ratios d'aspect (1:1, 2:1, 1:2...) pour couvrir toutes les formes d'objets possibles.

SSD: Single Shot MultiBox Detector

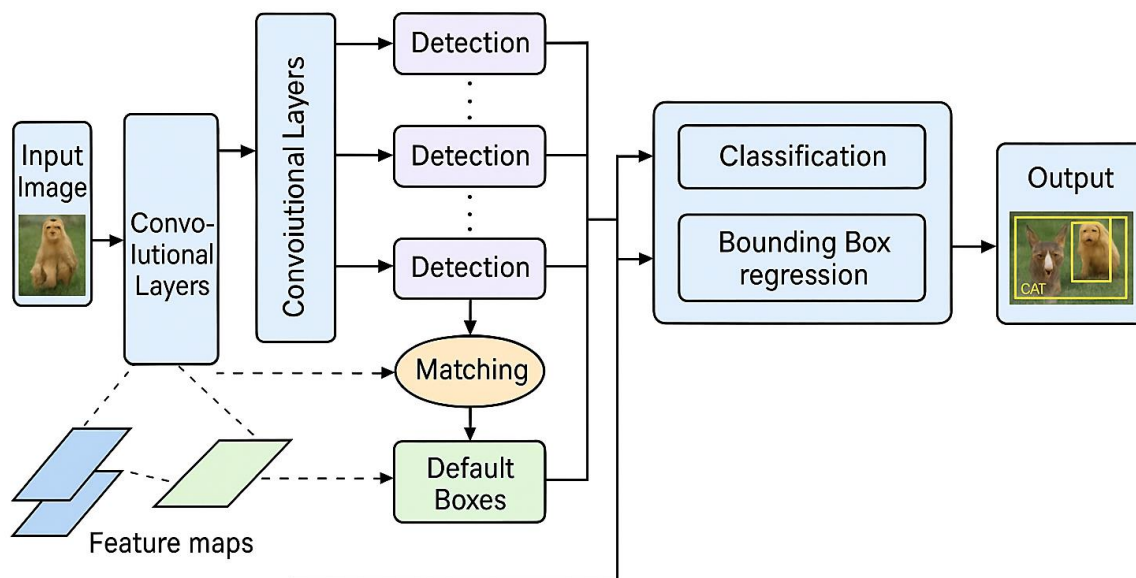


Figure 16: Architecture du modèle SSD (Single Shot MultiBox Detector)

Avantages

- Très **rapide**, adapté à la détection en temps réel (jusqu'à 45 FPS sur GPU).
- **Simplicité** d'implémentation par rapport aux méthodes à étapes multiples.
- Bonne performance sur des objets de **taille moyenne à grande**.

Limites

- Moins précis que Faster R-CNN pour les **petits objets** ou les scènes très encombrées.
- Peut générer plus de **faux positifs**, surtout si les ancres sont mal adaptées au jeu de données.

✓ YOLO [15]

YOLO (You Only Look Once) est une architecture de détection d'objets en une seule étape (*one-stage object detector*) introduite par Joseph Redmon en 2016. Elle repose sur le principe fondamental de traiter l'ensemble d'une image en une seule passe à travers un réseau neuronal convolutif, pour prédire à la fois les **positions (bounding boxes)** et les **classes d'objets**. Ce fonctionnement rapide et unifié en fait l'un des modèles les plus prisés dans des applications en temps réel.

Évolution des versions de YOLO

Depuis sa création, YOLO a connu de nombreuses évolutions :

- **YOLOv1–v3** : base solide avec détection rapide mais sensible aux petits objets.
- **YOLOv4** : intégration de techniques modernes (Mish, SPP, PANet), meilleures performances.
- **YOLOv5 (Ultralytics)** : non officiel mais très populaire ; modèle léger, exportable, modulaire.
- **YOLOv6/v7** : optimisés pour l'industrie avec précision et vitesse améliorées.
- **YOLOv8 (2023)** : support natif de la segmentation et tracking sans NMS.
- **YOLOv11 (2024)** : introduit un **backbone hybride CNN + Transformer**, des **têtes dynamiques**, et le support de **multi-tâches intégrées**, améliorant encore la précision tout en réduisant le coût computationnel (Ultralytics).

Fonctionnement général

YOLO divise l'image d'entrée en une **grille de cellules** (ex. 13×13). Chaque cellule est responsable de prédire un nombre fixe de **boîtes englobantes** avec :

- les coordonnées (x, y, largeur, hauteur),

- un score de confiance,
- et une probabilité de classe.

Les sorties sont ensuite filtrées via **NMS (Non-Max Suppression)** pour éliminer les prédictions redondantes.

Les **blocs de l'architecture YOLO** typique comprennent :

- **Backbone** (ex. CSPDarknet, EfficientNet) : extraction des caractéristiques visuelles,
- **Neck** (PANet, BiFPN) : fusion des informations multi-échelle,
- **Head** : prédiction de bounding boxes + classes.

YOLOv11 remplace certaines de ces étapes par des **têtes dynamiques** et des modules d'attention (PSA), supprimant parfois le besoin de NMS, tout en permettant un traitement **multi-tâches (segmentation, classification, suivi, pose estimation)**.

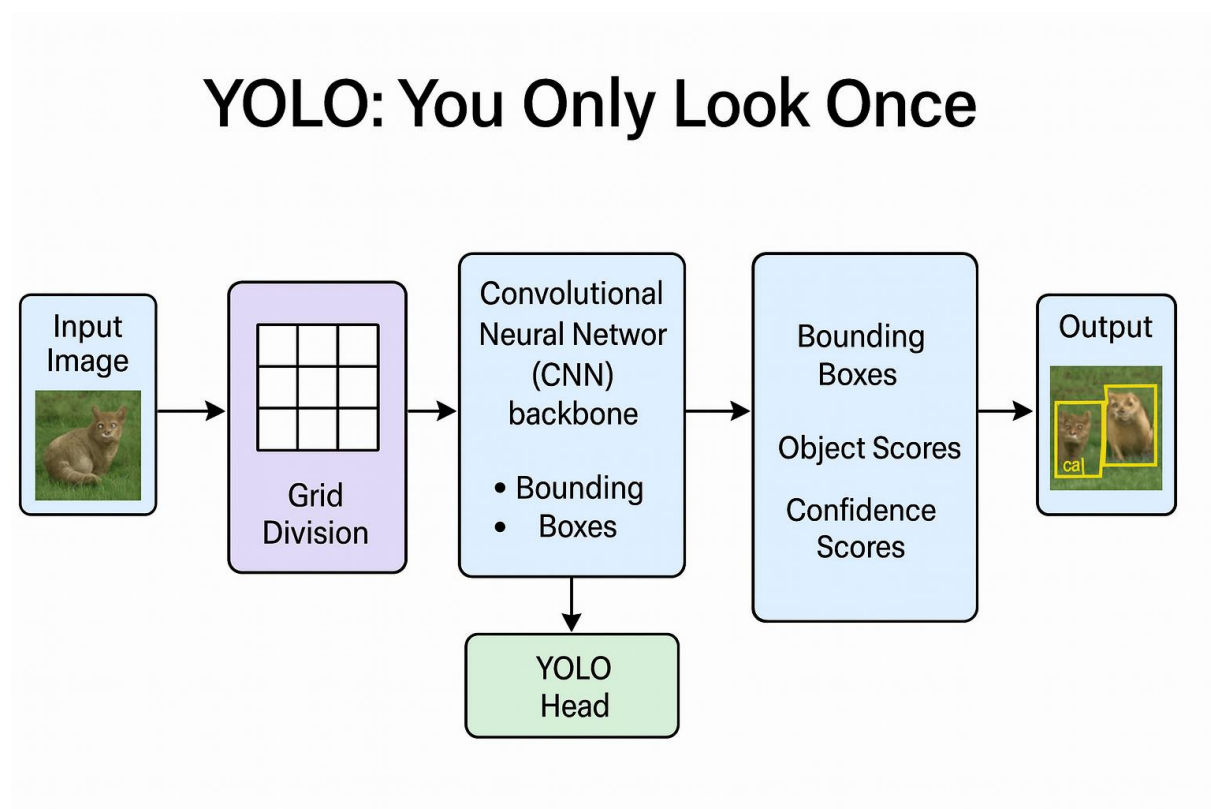


Figure 17: Architecture de YOLO Avantages de YOLO

- **Rapidité extrême** : parfaitement adapté aux applications temps réel.
- **Modularité** : existe en plusieurs tailles (nano, small, medium, large).
- **Précision renforcée** : notamment sur les dernières versions (v8, v11).
- **Support actif** : grande communauté, compatibilité avec ONNX, TensorRT, CoreML.
- **Adapté aux documents** : détecte facilement logos, signatures, cachets et structures visuelles.

Comparaison entre SSD et YOLO : deux approches one-stage en détection d'objets

La détection d'objets est un pilier de la vision par ordinateur, permettant d'identifier et de localiser plusieurs objets dans une même image.

Parmi les approches dites « **one-stage** », les modèles **SSD (Single Shot MultiBox Detector)** et **YOLO (You Only Look Once)** occupent une place centrale. Contrairement aux méthodes traditionnelles telles que **Faster R-CNN**, qui opèrent en deux étapes (proposition de régions, puis classification), ces deux modèles réalisent **simultanément la localisation et la classification** des objets. Ils se distinguent par leur **vitesse d'inférence** élevée et leur capacité à fonctionner **en temps réel**, ce qui les rend particulièrement attractifs pour des applications telles que la vidéo-surveillance, les systèmes embarqués, ou encore l'analyse automatique de documents visuels (factures, cachets, logos...).

Approche conceptuelle commune

SSD et YOLO partagent une philosophie fondamentale : ils opèrent en une seule étape de bout en bout (*end-to-end*). L'image est traitée par un réseau convolutif, qui produit directement des prédictions pour chaque cellule ou zone, incluant :

- Les coordonnées des boîtes englobantes (bounding boxes),
- Les scores de confiance (objectness),
- Et les classes d'objets détectés.

Cette approche évite l'**utilisation de réseaux de proposition de régions** (Region Proposal Networks), ce qui simplifie le flux de traitement, réduit le temps d'inférence, et diminue la complexité du modèle.

Différences techniques notables

Malgré leur similitude conceptuelle, SSD et YOLO diffèrent sur plusieurs aspects :

- **Stratégie multi-échelle :**
 - SSD exploite différentes couches de sa feature map pour effectuer des prédictions à plusieurs résolutions, améliorant ainsi sa capacité à détecter des objets de tailles variées.
 - YOLO, quant à lui, utilise des *necks* comme PANet ou BiFPN dans ses versions récentes pour assurer la fusion des caractéristiques à différentes échelles.
- **Division spatiale :**
 - YOLO divise l'image d'entrée en une grille régulière, chaque cellule prédisant plusieurs boîtes.
 - SSD s'appuie sur des ancres (anchor boxes) générées à différentes résolutions pour prédire des objets.
- **Backbone et modularité :**
 - SSD a initialement été conçu avec **VGG16**, puis adapté à d'autres réseaux légers comme **MobileNet** pour des applications embarquées.
 - YOLO a évolué pour intégrer des backbones plus puissants (**CSPDarknet**, **EfficientNet**, voire **transformers** dans YOLOv11), avec plusieurs tailles de modèles (nano, small, medium...).
- **Performances récentes :**
 - YOLOv11 intègre des innovations comme les **Dynamic Heads**, **PSA attention**, ou la **détection multi-tâches** native (détection + segmentation + classification).
 - SSD, bien que toujours utilisé, n'a pas connu de mise à jour majeure depuis SSD512, ce qui limite son évolution comparée à YOLO.

Tableau 5: Comparaison entre SSD et YOLO

Critère	YOLO (v5 à v11)	SSD
Vitesse	Très rapide (temps réel GPU et mobile)	Rapide (CPU-friendly, MobileNet-SSD)
Précision	Élevée, améliorée dans v8/v11	Moyenne à bonne selon les tailles d'objets
Modularité	Large éventail de versions et tâches (segmentation, tracking)	Détection simple uniquement

Support et maintenance	Très actif (Ultralytics, communauté GitHub)	Moins actif, peu de versions modernes
Utilisation documentaire	Très adapté (logos, tableaux, structures documentaires)	Valable si contraintes de ressources matérielles

Conclusion

En résumé, **YOLO** se positionne comme le **modèle le plus équilibré** entre **précision**, **vitesse**, et **polyvalence** pour des applications modernes de détection d'objets. Grâce à ses améliorations continues et à son support actif, il répond aux besoins variés d'un système intelligent d'analyse visuelle documentaire.

2.6.3. Fonctionnement technique des bibliothèques OCR

Chaque bibliothèque OCR adopte une approche algorithmique spécifique :

- **Tesseract** [16]: repose sur une architecture LSTM (Long Short-Term Memory). Il segmente les lignes et caractères, applique une reconnaissance caractère par caractère, et réalise une post-correction via un dictionnaire de langue. Il est efficace sur des documents simples bien scannés.
- **EasyOCR** [17]: combine **CNN** pour l'extraction de caractéristiques visuelles, **RNN (GRU)** pour la séquence temporelle des caractères, et **CTC (Connectionist Temporal Classification)** comme couche de décodage. Il produit des bounding boxes, détecte les lignes et lit chaque zone textuelle avec une grande flexibilité.
- **PaddleOCR** [18]: adopte une approche modulaire avancée :
 - **DBNet** pour la détection de zones textuelles (bounding boxes)
 - **ResNet/CNN** pour l'extraction de traits visuels
 - **Attention-based Encoder-Decoder** pour la reconnaissance
 - Support de structure de document avec **Layout Recovery**

Ces bibliothèques effectuent généralement un prétraitement de l'image (redressement, suppression de bruit), une détection de texte, puis une lecture avec modélisation séquentielle.

2.6.4. Limites et améliorations

3 .Développement d'une application Web

Le développement de la solution repose sur une architecture web complète, composée d'un **backend intelligent**, d'un **frontend interactif**, et d'un **environnement conteneurisé Docker**. Cette application a été conçue pour automatiser le traitement des factures à travers des modules d'intelligence artificielle orchestrés via une API performante et documentée.

3.1 un framework moderne pour la création d'API

FastAPI [19] est un framework web open-source écrit en Python, conçu pour créer rapidement des API performantes, évolutives et documentées. Il repose sur :

- les **annotations type hints** de Python (Pydantic),
- une exécution **asynchrone** (async/await),
- une documentation interactive générée automatiquement via **Swagger/OpenAPI**.

Avantages :

- Rapide à coder, facile à maintenir
- Performances élevées grâce à Starlette et Uvicorn
- Intégration directe avec les modèles IA (OCR, LLM, etc.)

3.1.1 Alternatives à FastAPI : Flask et Streamlit

- **Flask** : micro-framework minimaliste, stable et éprouvé, largement utilisé pour les backends API simples. Moins rapide que FastAPI mais très flexible.
- **Streamlit** [20]: framework orienté **interfaces utilisateur**, permet de créer rapidement des applications interactives de visualisation ou de démonstration IA sans HTML/JS. Moins adapté pour les API REST, mais idéal pour des **dashboards** internes.

Tableau 6: Comparaison entre flask ,streamlit et fast api

Framework	Usage principal	Avantages	Cas d'usage typique
FastAPI	API REST/ML	Rapide, typé, Swagger intégré	Backend IA, microservices

Flask	API REST généraliste	Simple, documentation vaste	Petites APIs, prototypage rapide
Streamlit	Interface IA/web interactive	Très visuel, pas de frontend	Dashboard OCR, démo LLM

3.2 Interface utilisateur basée sur Next.js

La conception d'interfaces web pour les systèmes intelligents modernes repose sur des frameworks capables d'allier **performance, interactivité et flexibilité**. Dans cette optique, **Next.js** [21] s'impose comme une solution robuste pour construire des interfaces réactives et modulables, notamment dans les projets intégrant des traitements IA lourds tels que l'OCR ou la détection d'objets.

Pourquoi Next.js est pertinent dans ce contexte ?

Next.js repose sur **React** et permet une combinaison efficace de rendu côté serveur (SSR), génération statique (SSG), et rendu dynamique. Cela garantit :

- un **chargement rapide** des pages,
- une **navigation fluide** sans rechargement complet,
- une **compatibilité parfaite avec les appels API REST** pour interagir avec des services de backend tels que FastAPI.

L'architecture de Next.js facilite aussi l'organisation des composants et l'adoption de bibliothèques modernes comme **Tailwind CSS** pour un design responsive et personnalisable.

Usage de templates d'interface préconçus

Dans les projets de traitement de documents, il est courant d'**utiliser ou adapter des modèles d'interfaces déjà conçus**, disponibles via des plateformes comme :

- **v0.dev (Vercel)** [22]: génération d'interfaces React par IA,

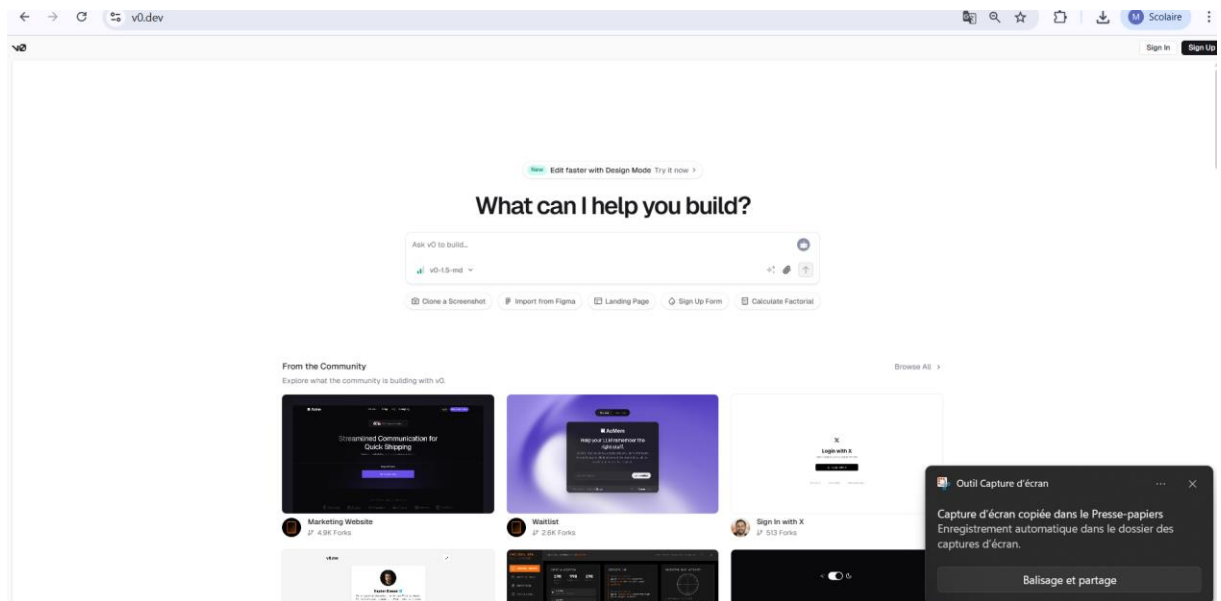


Figure 18: interface de V0

- **Hugging Face Spaces** : interfaces prêtes pour visualiser ou tester des modèles IA, ceci illustré par cette figure :

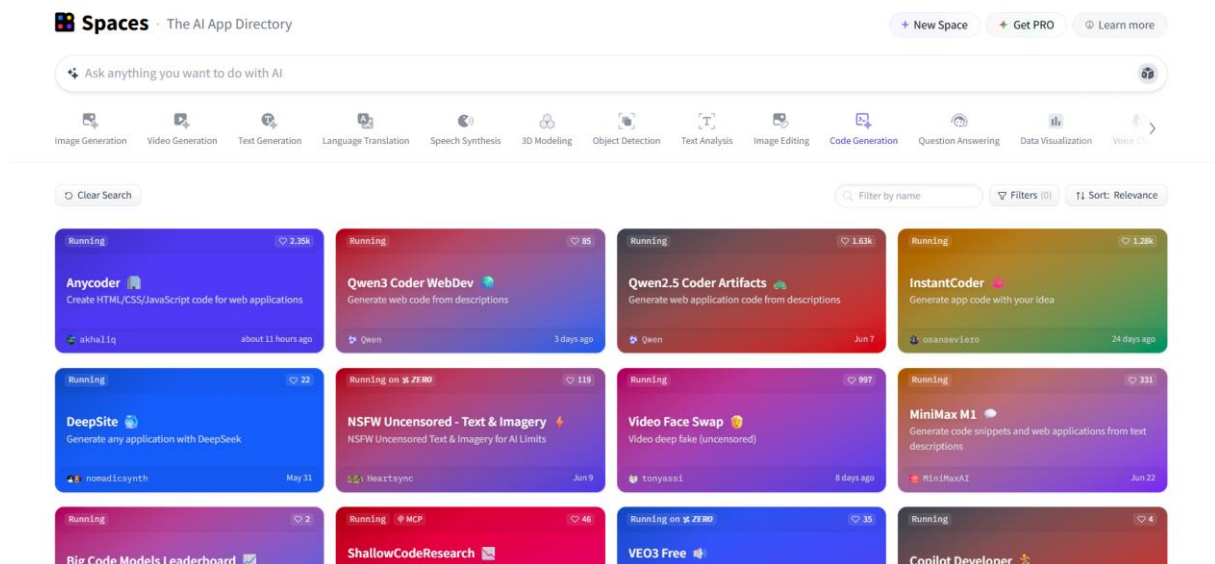


Figure 19 : Interface de Spaces pour ai app directory dans Hugging face

- **GitHub / Tailwind Components / shadcn/ui** : bibliothèques de composants stylisés.

Ces Templates offrent généralement :

- des modules d'**upload de documents** avec aperçu,
- des **indicateurs de traitement** (progression, erreurs, résultats),

- des **zones dynamiques de visualisation** (extraction de texte, tableaux, JSON structuré...).

Ils permettent de **réduire le temps de développement**, tout en assurant **une qualité visuelle et fonctionnelle élevée** adaptée aux cas d'usage IA (analyse de factures, lecture de documents, reconnaissance d'éléments visuels...).

Avantages concrets pour les applications IA documentaires

Dans un pipeline de traitement intelligent de documents, l'interface joue un rôle clé dans :

- la **collecte des entrées utilisateur** (téléversement de fichiers),
- la **visualisation instantanée des résultats extraits** (via tableaux, zones de texte ou cartes interactives),
- la **gestion d'états de réponse API** (chargement, erreur, succès),
- l'**interopérabilité entre le frontend et le backend** (via axios ou fetch vers les endpoints FastAPI).

Ainsi, Next.js, combiné à des templates d'interface adaptés, constitue une solution efficace pour concevoir des **interfaces documentaires intelligentes** où réactivité, ergonomie et modularité sont essentielles.

3.3 Conteneurisation et déploiement de l'application avec Docker

Le déploiement d'une application complète composée de plusieurs modules (API, interface utilisateur, traitements IA...) nécessite une approche qui garantit à la fois **portabilité**, **cohérence d'exécution**, et **facilité de mise en production**. C'est dans ce cadre que la solution **Docker** a été adoptée.

Qu'est-ce que Docker ? [23]

Docker est une plateforme open-source de conteneurisation qui permet d'exécuter des applications dans des **environnements isolés et standardisés** appelés conteneurs. Contrairement aux machines virtuelles classiques, les conteneurs partagent le noyau du système hôte, ce qui les rend **légers, rapides et portables**.

Chaque composant de l'application (API, frontend, services IA...) peut être encapsulé dans un conteneur, assurant un fonctionnement identique, quel que soit l'environnement (développement local, serveur cloud, machine de test...).

Structure de conteneurisation de l'application

Dans ce projet, trois grandes briques ont été conteneurisées :

1. Le backend Fast API (Python) :

- Inclut tous les modules IA nécessaires : Paddle OCR, YOLO, modèles LLM.
- Contient l'API REST accessible en local sur un port spécifique (ex. : 8000).

2. Le frontend Next.js (JavaScript/ React) :

- Sert l'interface utilisateur en mode développement ou compilation statique.
- Expose l'interface sur le port habituel (ex. : 3000).

3. Docker Compose :

- Outil de gestion multi-conteneurs qui permet de démarrer l'ensemble de l'application avec une seule commande (docker-compose up).
- Décrit tous les services, leurs dépendances, les ports exposés, et les volumes.

Avantages obtenus avec Docker

L'adoption de Docker dans le projet a apporté plusieurs bénéfices concrets :

Tableau 7: Avantages de Docker

Bénéfice	Détail
Portabilité	Même environnement sur tous les postes de développement
Isolation des services	Chaque composant tourne indépendamment, évitant les conflits
Facilité de déploiement	Lancement automatique de tous les services simultanément
Gain de temps	Plus besoin d'installer manuellement les dépendances locales
Test simplifié	Possibilité de tester l'ensemble du pipeline sur un seul poste

Exemple de scénario de déploiement

1. L'utilisateur clone le dépôt du projet.
2. Il lance docker-compose up à la racine.
3. Docker construit les images des conteneurs :
 - Le backend démarre et attend les requêtes API ;
 - Le frontend s'affiche via le navigateur ;
4. L'utilisateur peut alors téléverser une facture, lancer l'analyse et observer les résultats
— **sans aucune configuration manuelle.**

Conclusion de chapitre:

"Après avoir présenté les fondements théoriques sur lesquels repose notre travail, nous abordons dans le chapitre suivant l'architecture concrète de la solution proposée, ainsi que sa mise en œuvre technique à travers une chaîne de traitement complète et modulaire."

Chapitre 3

Architecture de la solution envisagée

Introduction:

"Ce chapitre constitue le pivot de notre travail, détaillant de manière exhaustive la conception, l'implémentation et l'évaluation de la solution proposée pour l'automatisation du traitement des factures fournisseurs. L'environnement de développement et les bases théoriques ayant été abordés dans le Chapitre 1 et 2.

ce chapitre se concentrera sur les aspects pratiques et les résultats. Nous y aborderons successivement l'architecture globale du système, la méthodologie rigoureuse de préparation et d'annotation des données, ainsi que la mise en œuvre et les performances de chacun des modules constitutifs : la classification des documents, la détection d'objets avec YOLOv8 et l'extraction structurée d'informations textuelles. Enfin, nous décrirons l'intégration de ces composants au sein d'une API unifiée."

1 Architecture Globale de la Solution

La solution que nous avons développée adopte une architecture modulaire, permettant un traitement séquentiel et robuste des factures fournisseurs, de l'image initiale aux données structurées. Le processus global est orchestré à travers plusieurs modules interconnectés, chacun étant spécialisé dans une tâche spécifique.

ARCHITECTURE GLOBALE DE LA SOLUTION

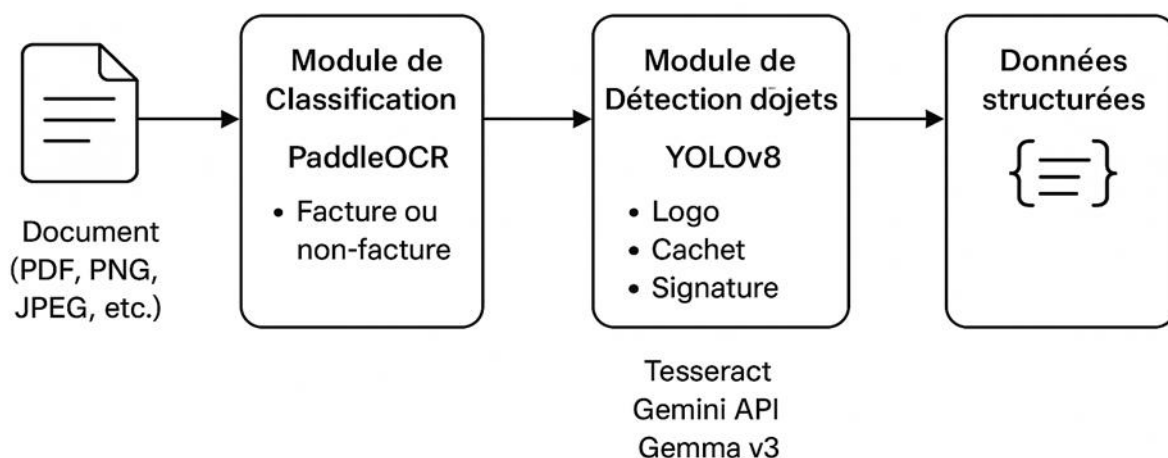


Figure 20: Schéma de l'architecture globale de la solution pipeline

✓ Pipeline de la solution:

Le flux de traitement débute par l'ingestion d'un document facture. Ce document est d'abord acheminé vers le **Module de Classification**, dont le rôle est d'identifier et de catégoriser le type de facture pour orienter les traitements ultérieurs. Ce module s'appuie sur l'extraction de texte via **PaddleOCR** combinée à une analyse de **mots-clés** pertinents pour attribuer la catégorie appropriée au document. Une fois le type de document déterminé, l'image est transmise au **Module de Détection d'Objets**. Ce module, basé sur l'architecture de Deep Learning **YOLOv8**, est spécifiquement entraîné pour localiser et délimiter les zones d'intérêt critiques sur la facture, telles que les logos ou les cachets, les signatures.

Ensuite on passe au **Module d'Extraction de Texte**. Ce module combine la reconnaissance optique de caractères (OCR) via des moteurs **OCR comme Tesseract** pour la conversion image-texte, avec l'intelligence des grands modèles de langage comme **Gemini** ou **Gemma** pour extraire et structurer les informations clés de manière précise. Enfin, les données structurées et validées sont générées, prêtes à être intégrées dans les systèmes d'information.

Cette approche modulaire assure une flexibilité, une maintenabilité et une évolutivité accrues de l'ensemble du système, permettant un traitement efficace des factures fournisseurs hétérogènes.

Mais avant d'entamer cette approche, il existe une étape cruciale Compréhension et préparation des données (*Data Understanding & Data Preparation*)

Section1: Compréhension et préparation des données (*Data Understanding & Data Preparation*)

Introduction

Dans le cadre de la méthodologie CRISP-DM, la compréhension et la préparation des données constituent deux phases fondamentales qui conditionnent la qualité des résultats obtenus lors des étapes de modélisation et d'évaluation. Cette section présente les actions entreprises pour analyser la nature des documents à traiter, évaluer leur qualité, puis préparer ces données de manière rigoureuse afin de garantir un apprentissage optimal des modèles d'intelligence artificielle.

1. Compréhension des données (Data Understanding)

1.1 Origine et nature des données

Le jeu de données mis à disposition par l'entreprise comprend 11 851 documents, localisés dans un répertoire intitulé LISTE DES FACTURES. Ces documents sont issus de la gestion réelle des factures fournisseurs dans le module Trésorerie du logiciel OMRAN Estate. Ils sont le reflet des conditions opérationnelles variées (fournisseurs multiples, formats hétérogènes, qualité de scan différente).

Répartition par format :

- ✓ PDF : environ 80 % du total, générés par des scanners ou imprimantes.
- ✓ Images (JPEG, PNG) : environ 15 %, issues principalement de captures mobiles ou scanners à plat.
- ✓ Fichiers tableur (XLS, XLSX) : environ 5 %, représentant des synthèses ou factures exportées.

1.2 Objectifs de l'analyse exploratoire

- Identifier la typologie des documents (factures réelles, reçus, duplicatas, etc.).
- Évaluer la qualité graphique des documents (résolution, netteté, orientation).
- Repérer les variations de structure entre fournisseurs (emplacement des champs, formats de date, devises).
- Détecter les éléments visuels caractéristiques (logo, cachet, signature).
- Quantifier les anomalies potentielles (documents incomplets, bruit visuel, doublons).

1.3 Analyse qualitative et statistique

Un échantillon de 500 documents a été sélectionné pour une analyse semi-automatisée, Les résultats ont révélé :

Tableau 8 :statistiques

Caractéristique observée	Fréquence estimée
Présence d'un logo fournisseur	~ 74 %
Présence d'un cachet ou signature	~ 65 %
Documents flous ou inclinés	~ 12 %
Format de date non standard	~ 18 %
Documents non pertinents (ex. reçus)	~ 9 %
Doublons (documents identiques répétés)	~ 6 %

1.4 Diversité structurelle et linguistique

Les documents proviennent de divers fournisseurs, ce qui entraîne une variabilité forte dans :

- La position des champs clés (en-tête, pied de page, colonnes latérales) ;
- Les types de formats numériques : HT / TTC séparés ou combinés, devises multiples (DT, EUR) ;
- Les langues utilisées : la majorité des documents sont en français, mais certains contiennent des mentions bilingues ou symboliques en arabe ou en anglais.

Cette diversité a un impact direct sur la capacité de généralisation des modèles d'extraction et nécessite une stratégie d'apprentissage adaptée.

1.5 Contraintes spécifiques rencontrées

Tableau 9:contraintes

Problème identifié	Impact potentiel
Documents scannés flous ou inclinés	Mauvaise reconnaissance OCR
Présence de doublons	Biais dans l'apprentissage
Variabilité de structure	Difficulté de normalisation
Données multilingues ou symboliques	Échec possible du modèle OCR
Présence de documents non pertinents	Bruit dans le dataset

1.6 Conclusion

La phase de compréhension des données a permis d'acquérir une vision claire de la nature et de la complexité des documents à traiter. Elle a mis en lumière les défis liés à la qualité visuelle, à l'hétérogénéité des structures, aux langues utilisées et aux éléments visuels présents dans les factures.

Ces observations justifient la mise en place d'un pipeline intelligent intégrant des modèles d'OCR, de vision par ordinateur et de traitement du langage naturel. Les résultats obtenus dans cette phase fournissent les fondations nécessaires à la phase de préparation des données, qui consistera à organiser, annoter et transformer les documents en vue de l'entraînement de modèles performants.

2. Préparation des données

La phase de préparation des données a pour objectif de transformer les documents bruts en un format exploitable par les modèles d'intelligence artificielle, tout en assurant leur qualité et leur cohérence.

2.1 Normalisation des formats de documents

La première étape de préparation a consisté à homogénéiser les formats des documents reçus. Le corpus initial comprenait une variété de fichiers : PDF, JPEG, PNG, ainsi que quelques fichiers Excel. Pour faciliter le traitement automatique et garantir la compatibilité avec les modules d'analyse visuelle (OCR, détection d'objets), l'ensemble des documents a été converti en format image (.png ou .jpeg).

Cette standardisation présente plusieurs avantages :

- Lecture uniforme par les moteurs OCR (PaddleOCR, Tesseract)
- Compatibilité avec les outils de détection d'objets (YOLOv8)
- Simplification des étapes d'annotation et de visualisation

La conversion a été réalisée à l'aide de bibliothèques Python telles que **pdf2image**, **PIL** et **OpenCV**. Un redimensionnement standard a été appliqué pour uniformiser les dimensions, et une première conversion en niveaux de gris a été effectuée pour réduire la complexité des données.

```
import cv2
import pytesseract
from PIL import Image
import os
```

Figure 21: les bibliothèques

Afin de préparer davantage les images à l'entraînement via des réseaux de neurones convolutionnels (CNN), une série de transformations normalisées a également été appliquée en amont des modèles d'apprentissage :

- Redimensionnement à 224×224 pixels (pour l'entraînement des CNN)
- Conversion en tenseur pour traitement GPU avec PyTorch
- Normalisation avec les moyennes et écarts types d'ImageNet.

```
# === 2. Prétraitement ===
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

dataset = datasets.ImageFolder(train_dir, transform=data_transforms)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)
```

Figure 22: capture d'écran illustrant le prétraitement

2.2 Amélioration de la Qualité des Images par Prétraitement

Au-delà de la normalisation des formats, une attention particulière a été portée à l'amélioration de la qualité intrinsèque des images. En effet, la présence de bruit, de flou, d'ombres ou de contrastes faibles dans les documents scannés ou photographiés peut fortement impacter la performance des moteurs d'OCR et la précision des modèles de détection. Pour pallier ces problèmes, une chaîne de prétraitement avancée a été mise en œuvre au sein de l'extracteur OCR optimisé (OptimizedOCRExtractor).

Ces étapes visent à :

- **Réduire le bruit et les artefacts** : Éliminer les imperfections visuelles qui pourraient être interprétées comme du texte ou brouiller les caractères.
- **Améliorer la clarté et le contraste du texte** : Rendre les caractères plus nets et plus distincts du fond.
- **Préparer l'image pour une binarisation optimale** : Faciliter la conversion en une image purement noir et blanc, essentielle pour la reconnaissance.

Les principales techniques appliquées incluent :

1. Conversion en Niveaux de Gris (Grayscale) :

- Après la conversion initiale si nécessaire, l'image est uniformément traitée en niveaux de gris, réduisant les canaux de couleur à un seul canal d'intensité lumineuse. Ceci simplifie les calculs tout en conservant l'information pertinente pour la reconnaissance des formes des caractères.

```
def preprocess_image(self, image_path: str) -> str:
    """
    Préprocesse l'image pour améliorer l'OCR

    Args:
    |   image_path: Chemin vers l'image

    Returns:
    |   Chemin vers l'image prétraitée
    """
    try:
        # Charger et valider l'image
        image = cv2.imread(image_path)
        if image is None:
            raise ValueError(f"Impossible de charger l'image: {image_path}")

        # Vérifier les dimensions
        height, width = image.shape[:2]
        if height < 10 or width < 10:
            print(f"Image trop petite pour préprocessing: {width}x{height}")
            return image_path

        # Convertir en niveaux de gris
        if len(image.shape) == 3:
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        else:
            gray = image
```

Figure 23: validation ,vérification, conversion en gris des images

2. Filtrage Bilatéral (Bilateral Filtering) :

- Cette technique de dé-bruitage est utilisée pour lisser les zones de l'image sans estomper les bords importants. Contrairement aux filtres de lissage classiques qui peuvent rendre le texte flou, le filtre bilatéral préserve les contours des lettres et des chiffres, ce qui est crucial pour maintenir la netteté des caractères.

```
# Appliquer un filtre pour supprimer le bruit (seulement si l'image est assez grande)
if height > 50 and width > 50:
    gray = cv2.bilateralFilter(gray, 11, 17, 17)
```

Figure 24: filtrage bilatéral

3. Binarisation Adaptative (Adaptive Thresholding) :

- Suite au dé-bruitage, l'image en niveaux de gris est convertie en une image binaire (noir pur pour le texte, blanc pur pour le fond). La binarisation adaptative est particulièrement efficace car elle calcule un seuil de conversion différent pour chaque petite région de l'image. Cela permet de gérer les variations d'éclairage, les ombres ou les fonds non uniformes souvent rencontrés sur les documents numérisés, garantissant une séparation nette entre le texte et l'arrière-plan.

Ces étapes de prétraitement sont intégrées dans le pipeline d'extraction, assurant que le moteur d'OCR et les modèles de vision par ordinateur opèrent sur des images de la meilleure qualité possible, maximisant ainsi la précision d'extraction et de classification des données.

```
# Binarisation adaptative pour de meilleurs résultats
try:
    # Essayer d'abord la binarisation adaptative
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
except Exception:
    # Si ça échoue, utiliser la binarisation simple
    _, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)

# Vérifier le résultat de la binarisation
if thresh is None or thresh.size == 0:
    print("Erreur lors de la binarisation, utilisation de l'image originale")
    return image_path

# Sauvegarder l'image prétraitée
base_name = os.path.splitext(image_path)[0]
processed_path = f"{base_name}_optimized_processed.png"

success = cv2.imwrite(processed_path, thresh)
if not success:
    print("Erreur lors de la sauvegarde de l'image prétraitée")
    return image_path

return processed_path
```

Figure 25: Script python de Binarisation Adaptative

2.3 Regroupement par lots de traitement (Batching)

Après la conversion des documents sources (PDF, Excel, etc.) en images .png et .jpeg, une étape essentielle a consisté à organiser ces fichiers de manière structurée pour faciliter leur traitement. Pour cela, un script Python a été développé afin de **répartir automatiquement les images dans des lots de 500 fichiers**. Cette segmentation vise à :

- réduire la complexité lors de l'annotation manuelle ou semi-automatique,
- répartir uniformément les données pour l'apprentissage,
- améliorer la traçabilité des étapes de traitement.

Chaque image a été classée dans un sous-dossier portant un nom standardisé (par exemple, batch_001, batch_002, etc.). Ce processus permet de diviser efficacement un corpus volumineux en sous-ensembles gérables tout en assurant une organisation logique des données.

```
import shutil

INPUT_DIR = r"C:\Users\saada\Documents\IMAGES"
BATCH_SIZE = 500
OUTPUT_DIR = os.path.join(INPUT_DIR, "batches")

os.makedirs(OUTPUT_DIR, exist_ok=True)

images = [f for f in os.listdir(INPUT_DIR) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]

for idx, img in enumerate(images):
    batch_num = idx // BATCH_SIZE + 1
    batch_dir = os.path.join(OUTPUT_DIR, f"batch_{batch_num:03}")
    os.makedirs(batch_dir, exist_ok=True)
    shutil.move(os.path.join(INPUT_DIR, img), os.path.join(batch_dir, img))

print("[✅] Images déplacées par lots dans :", OUTPUT_DIR)
```

[✅] Images déplacées par lots dans : C:\Users\saada\Documents\IMAGES\batches

Figure 26: – Script Python de regroupement automatique des images en lots de 500

2.4 Vérification de la qualité -conversion :

Une vérification manuelle a été menée après la conversion et la segmentation par lots, afin de s'assurer de la lisibilité des fichiers convertis, de leur intégrité et de la répartition équilibrée des

documents dans chaque batch. Les documents de qualité insuffisante (flous, tronqués, fortement bruités) ont été marqués pour exclusion ou retraitement.

2.5 Annotation des données pour la détection d'objets (YOLOv8)

L'annotation des données a été réalisée à l'aide de l'outil **LabelImg** [24], une interface graphique libre permettant de dessiner manuellement des boîtes englobantes (bounding boxes) sur les images.

Voici le pipeline adopté pour constituer un dataset prêt à l'entraînement de YOLOv8 :

1. Installation de LabelImg :

L'outil a été installé localement (via pip ou GitHub). Il permet de charger un répertoire contenant des images .jpg ou .png.

2. Chargement du dataset :

On ouvre le dossier images/train ou images/val depuis LabelImg. Les images s'affichent une à une dans l'interface.

3. Création des annotations :

Pour chaque image :

- On encadre manuellement les éléments visuels d'intérêt (ex. logo, cachet) avec l'outil de sélection.
- Chaque zone annotée se voit attribuer une **classe** (ex. 0 pour "logo", 1 pour "cachet").
- Les annotations sont enregistrées au **format YOLO**, soit un fichier .txt par image contenant les coordonnées normalisées des objets.

4. Organisation automatique :

Les fichiers .txt sont sauvegardés dans un dossier labels/ parallèle à images/. L'ensemble respecte donc la structure YOLO suivante :

5. Vérification finale :

Un script Python a permis de filtrer les fichiers problématiques (images sans labels ou mal annotées) et de s'assurer de la cohérence totale des paires image + annotation.

Ce processus rigoureux garantit que chaque image dispose d'un fichier d'annotation compatible avec l'entraînement YOLOv8, facilitant ainsi la phase de modélisation.

Alors

Mise en pratique ;

Dans le but d'entraîner un modèle YOLOv8 capable de détecter automatiquement les éléments visuels présents sur les factures (notamment les logos et les tampons/cachets), une phase d'annotation supervisée a été mise en œuvre. Pour cela, une nouvelle version du dataset a été reconstruite à partir des images originales, sous le nom `dataset_rebuilt` (extraite d'après batch 002)

Le dataset, nommé `dataset_rebuilt`, a été structuré selon la norme YOLO :

```
dataset_rebuilt/
├── images/
│   ├── train/
│   ├── val/
│   └── test/
├── labels/
│   ├── train/
│   ├── val/
│   └── test/
```

Les annotations ont été réalisées avec LabelImg. Chaque fichier .txt associé à une image contient les coordonnées des objets détectés, au format YOLO (classe, x_center, y_center, width, height).

- Répartition du dataset

Le jeu de données a été réparti comme suit :

Tableau 10: jeu de données

Partition	Nombre d'images	Pourcentage
Train	249	~70 %
Validation	70	~20 %

Test	35	~10 %
------	----	-------

Afin d'organiser les images annotées dans une structure conforme au format YOLOv8, un script Python a été développé pour automatiser la répartition des données en deux ensembles distincts : 80 % pour l'entraînement et 20 % pour la validation. Ce script effectue plusieurs opérations critiques :

- Il crée automatiquement les dossiers nécessaires (train/images, train/labels, val/images, val/labels) si ceux-ci n'existent pas.
- Il copie les images ainsi que leurs fichiers d'annotation .txt correspondants dans les bons répertoires.
- Il effectue une vérification systématique : si une annotation est absente ou si un fichier a été préalablement identifié comme à ignorer (dans le cas d'annotations défectueuses ou bruitées), celui-ci est automatiquement exclu du dataset.

Cette étape garantit une organisation rigoureuse du corpus tout en éliminant les cas incohérents ou problématiques.

```

3]: import os

# Répertoires des images et annotations
images_dir = r'C:\Users\saada\Documents\IMAGES\batches\batch_002\factures'
labels_dir = r'C:\Users\saada\Desktop\annotations'
train_images_dir = r'C:\Users\saada\Documents\YOLO\train\images'
val_images_dir = r'C:\Users\saada\Documents\YOLO\val\images'

# Création des répertoires si nécessaires
os.makedirs(train_images_dir, exist_ok=True)
os.makedirs(val_images_dir, exist_ok=True)

# Diviser les fichiers d'images en train et validation (88% train, 28% val)
image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg') or f.endswith('.png')]
split_index = int(0.8 * len(image_files))

train_files = image_files[:split_index]
val_files = image_files[split_index:]

# Créer le fichier train.txt
with open(r'C:\Users\saada\Documents\YOLO\train.txt', 'w') as train_txt:
    for f in train_files:
        train_txt.write(os.path.join(train_images_dir, f) + '\n')

# Créer le fichier val.txt
with open(r'C:\Users\saada\Documents\YOLO\val.txt', 'w') as val_txt:
    for f in val_files:
        val_txt.write(os.path.join(val_images_dir, f) + '\n')

# Créer le fichier obj.names (noms des classes)
classes = ['logo', 'cachet', 'signature', 'cachet+signature'] # Liste des classes
with open(r'C:\Users\saada\Documents\YOLO\obj.names', 'w') as names_file:
    for class_name in classes:
        names_file.write(class_name + '\n')

# Créer le fichier obj.data
with open(r'C:\Users\saada\Documents\YOLO\obj.data', 'w') as data_file:
    data_file.write(f'classes = {len(classes)}\n')
    data_file.write(f'train = C:/Users/saada/Documents/YOLO/train.txt\n')
    data_file.write(f'valid = C:/Users/saada/Documents/YOLO/val.txt\n')
    data_file.write(f'names = C:/Users/saada/Documents/YOLO/obj.names\n')
    data_file.write(f'backup = C:/Users/saada/Documents/YOLO/backup\n')

print("Les fichiers de configuration ont été créés avec succès.")

```

Les fichiers de configuration ont été créés avec succès.

Figure 27: Script Python de séparation et filtrage automatique des données pour l'entraînement et la validation.

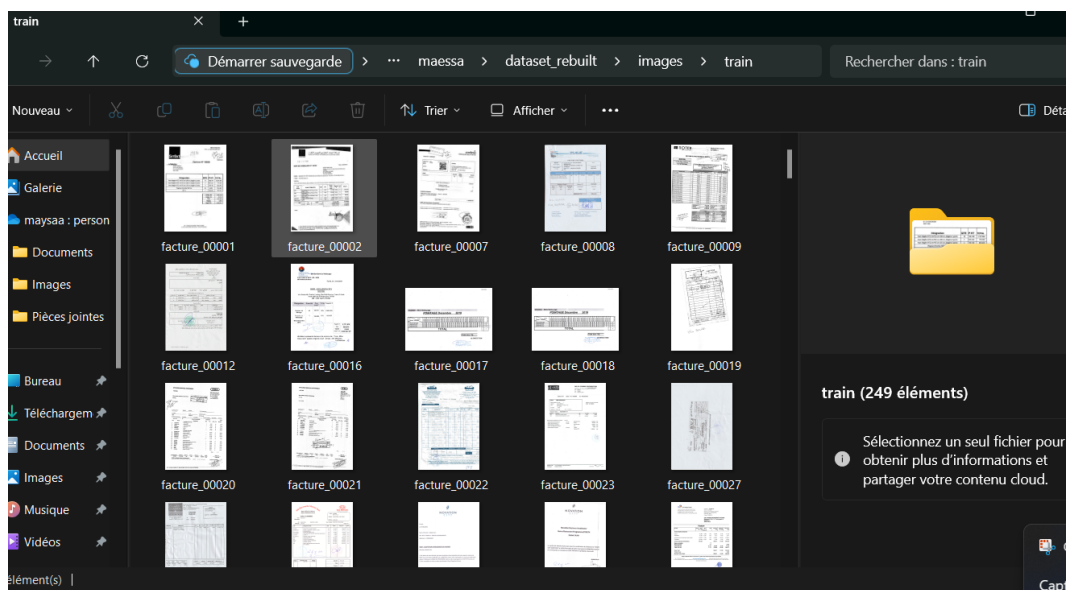


Figure 28: Exemple de répertoire 'train' du dataset YOLOv8 contenant des factures annotées.

Application dans l'outil labellmg [24] :

- Les dossiers images/ contiennent les factures au format .jpg ou .png. Les dossiers labels/ contiennent les fichiers .txt avec les annotations correspondantes pour chaque image.
- L'outil Labellmg a été utilisé pour annoter manuellement les objets visuels à détecter. Voici les étapes réalisées :
- - Chargement du dataset : les répertoires images/train, images/val et images/test ont été fournis à Labellmg.
- ✓ - Annotation des objets : pour chaque image, deux types de zones ont été encadrées :
 - Classe 0 → Logo
- ✓ - Classe 1 → Cachet / tampon
- ✓ - Sauvegarde automatique : chaque image annotée génère un fichier .txt dans le répertoire labels/ correspondant.
- ✓ - Répétition du processus pour l'ensemble des images du dataset reconstruit.

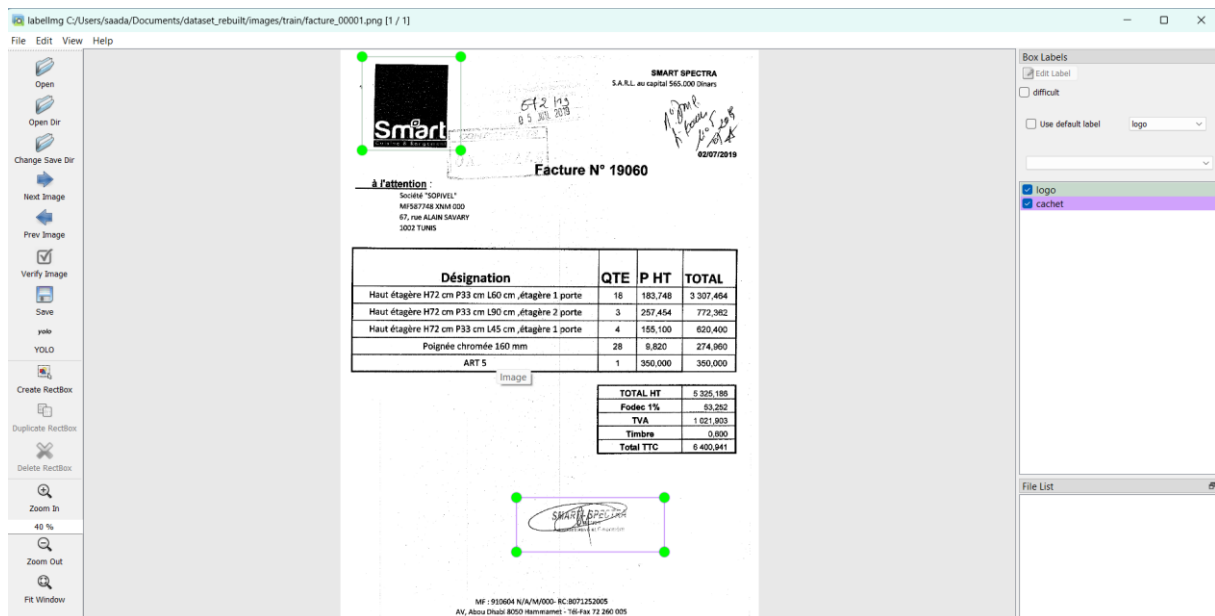


Figure 29: Annotation manuelle avec Labellmg sur une facture : encadrement du logo (classe 0) et du cachet (classe 1).

Chaque image annotée donne lieu à un fichier .txt contenant les coordonnées normalisées des objets détectés.

Par exemple, la capture ci-dessous illustre un fichier YOLO généré par Labellmg :

- La première ligne représente un logo (classe 0)
- La deuxième ligne correspond à un cachet (classe 1)

```
0 0.533529 0.074545 0.088235 0.050000
1 0.481765 0.382500 0.132941 0.122273
```

Figure 30: – Exemple de fichier d'annotation YOLOv8 généré après annotation manuelle.

Une fois les images et leurs annotations correctement organisées, **la dernière étape de préparation** consiste à créer le fichier de configuration data.yaml, indispensable pour l'entraînement du modèle YOLOv8.

- Ce fichier joue un rôle central : il indique au modèle l'emplacement des ensembles de données (train et val), ainsi que la liste complète des classes à détecter. Dans notre cas, seules deux classes sont effectivement utilisées (le logo et le cachet), mais d'autres classes génériques peuvent être présentes si le modèle est réutilisé ou étendu.
- Le fichier data.yaml a été généré automatiquement à l'aide d'un script Python utilisant la bibliothèque yaml. Ce script encode les chemins relatifs vers les images d'entraînement et de validation, le nombre total de classes, ainsi que leurs noms. Il est ensuite sauvegardé dans le répertoire de travail de YOLO.

```

]: import yaml

# Chemins vers Les répertoires d'images d'entraînement et de validation
train_images_dir = r'C:/Users/saada/Documents/YOLO/train/images'
val_images_dir = r'C:/Users/saada/Documents/YOLO/val/images'

# Liste complète des classes
data = {
    'train': train_images_dir,
    'val': val_images_dir,
    'nc': 19, # Nombre de classes (ici 19)
    'names': [
        'dog', 'person', 'cat', 'tv', 'car', 'meatballs', 'marinara sauce', 'tomato soup',
        'chicken noodle soup', 'french onion soup', 'chicken breast', 'ribs', 'pulled pork',
        'hamburger', 'cavity', 'logo', 'cachet+signature', 'signature', 'cachet'
    ] # Liste des 19 classes
}

# Chemin où le fichier data.yaml sera enregistré
data_yaml_path = r'C:/Users/saada/Documents/YOLO/data.yaml'

# Créer et sauvegarder le fichier data.yaml
with open(data_yaml_path, 'w') as f:
    yaml.dump(data, f, default_flow_style=False)

print(f"Le fichier data.yaml a été créé et sauvegardé dans {data_yaml_path}")

```

Le fichier data.yaml a été créé et sauvegardé dans C:/Users/saada/Documents/YOLO/data.yaml

Figure 31:– Exemple de fichier d'annotation YOLOv8 généré après annotation manuelle.



```

1 train: /content/drive/MyDrive/dataset/dataset_reduit/images/train
2 val: /content/drive/MyDrive/dataset/dataset_reduit/images/val
3 nc: 2
4 names: ['logo', 'cachet']
5

```

Figure 32: fichier data.yaml**Constatation :**

Dans le cadre de ce projet, l'absence de jeux de données annotés spécifiquement adaptés au cas d'usage – malgré une exploration approfondie de plateformes spécialisées telles que Kaggle et Roboflow – m'a conduit à créer un **dataset personnalisé** à partir de documents réels.

Plutôt qu'une contrainte, cette situation s'est révélée une **opportunité stratégique**, permettant de constituer un corpus parfaitement représentatif des documents effectivement rencontrés : factures fournisseurs hétérogènes, parfois floues, inclinées, multilingues ou mal scannées.

Pour bâtir ce jeu de données, j'ai mis en place un **pipeline d'annotation semi-automatisé**, combinant :

- **LabellImg**, une interface graphique intuitive, pour l'annotation manuelle des entités visuelles cibles (logos, cachets) ;
- et des **scripts Python sous Jupyter Notebook**, destinés à organiser les fichiers, répartir les images entre les ensembles d'entraînement, validation et test, et valider la cohérence des annotations.

Ce processus a permis d'assurer à la fois **la précision des annotations** et **l'efficacité du traitement en lot**, deux critères essentiels pour entraîner un modèle robuste.

Le résultat est un dataset conforme aux standards de la bibliothèque **YOLOv8** :

- fichiers d'annotations au format .txt (une ligne par objet : classe + coordonnées normalisées),
- structure arborescente classique (/train, /val, /test),
- fichier data.yaml généré dynamiquement, facilitant l'appel à la commande d'entraînement.

En résumé, cette phase préparatoire – bien qu’en partie manuelle – a permis de **poser des bases solides et réalistes** pour l’apprentissage automatique, dans un contexte caractérisé par une forte variabilité documentaire et l’absence de données labellisées prêtes à l’emploi.

Section 2 Entraînement et exécution du pipeline d’analyse

Cette section détaille le fonctionnement de chaque module clé composant l’architecture de la solution, en expliquant leurs rôles spécifiques et les technologies sous-jacentes.

1. Module de Classification des Documents (Facture vs Non-Facture)

Dans tout système de traitement automatique de documents, la première étape critique consiste à **déterminer la nature du document**. Dans notre cas, il s’agit de distinguer si le document est une **facture fournisseur** ou non.

Cette classification a une **valeur stratégique** :

- Elle **évite de mobiliser inutilement des ressources** d’analyse sur des documents non pertinents (devis, bons de commande, publicités, etc.)
- Elle **prévient les erreurs** dans les étapes de structuration des données
- Elle **accélère les traitements** en élaguant le pipeline dès l’amont

La classification sert donc de **point de décision initial** dans l’orchestration globale.

1.1 Objectif opérationnel

L’objectif de ce module est de répondre automatiquement à la question suivante :

Le document soumis est-il une facture ou non ?

En fonction de la réponse :

- **✓ Oui (Facture)** → Le document est transféré vers les modules d’extraction (visuelle, textuelle, sémantique)
- **✗ Non (Document sans valeur ajoutée)** → Il est soit :

- Rejeté
- Archivé pour audit
- Redirigé vers une file de validation manuelle

Ce tri permet de **canaliser intelligemment les flux de documents** dans un contexte de traitement de masse.

1.2 Technologies et Méthodes Utilisées

- PaddleOCR et PPChatOCRv4Doc

Le système exploite la puissance de la bibliothèque PaddleOCR, réputée pour son efficacité en reconnaissance optique de caractères. Plus précisément, il utilise le pipeline PPChatOCRv4Doc, une extension de PaddleOCR intégrant une compréhension avancée de la structure des documents (layout analysis), indispensable pour analyser des documents complexes comme les factures.

Étapes de traitement :

1. Détection des zones de texte :

Le modèle détecte précisément les régions contenant du texte dans le document, en dessinant des boîtes englobantes autour de ces zones.

2. Reconnaissance du texte :

Chaque région détectée est ensuite convertie en texte lisible par une couche de reconnaissance.

3. Analyse de la mise en page (layout) :

PPChatOCRv4Doc analyse les relations spatiales entre blocs de texte, images, tableaux, titres, permettant de reconstituer la structure logique du document (par ex. en-têtes, colonnes, sections totaux).

1.3 Logique de Classification

Critères utilisés

Pour décider si un document est une facture, le système analyse deux grandes catégories d'indices :

- **Indices sémantiques (mots-clés)**

Le module recherche la présence et la fréquence de termes spécifiques aux factures, tels que : *facture, invoice, total, montant, TVA, HT, TTC, prix unitaire, référence, date d'émission, fournisseur, client...*

Ces mots-clés sont des marqueurs forts d'une facture.

- **Indices structurels (mise en page)**

L'analyse de la structure du document vise à détecter des éléments récurrents dans les factures, notamment :

- Un en-tête avec les coordonnées du fournisseur et du client.
- Un tableau listant les articles, leurs quantités, prix unitaires, totaux.
- Une section récapitulative avec les montants totaux, taxes, etc.

1.4 Processus de décision

Les indices sont combinés selon une logique définie, qui peut être une règle simple (par exemple, "si au moins 3 mots-clés spécifiques sont présents et un bloc numérique important est détecté, alors document = facture") ou un modèle de classification binaire entraîné sur des exemples annotés.

- Score de Confiance (Confidence Score)

Définition

Le module génère un score quantitatif, appelé score de confiance, indiquant la probabilité que le document soit une facture. Ce score normalisé entre 0 et 1 permet d'évaluer la fiabilité de la classification.

Facteurs entrant dans le calcul :

1. **Pertinence des mots-clés détectés**

Chaque mot-clé identifié contribue positivement au score. Leur poids varie selon leur importance (ex: “numéro de facture” a plus de poids que “date”). La qualité de reconnaissance OCR (avec son propre score interne) est aussi prise en compte.

2. Conformité de la mise en page

Le système vérifie que la structure détectée correspond aux standards habituels d’une facture (en-tête, tableau, totaux), ce qui renforce la confiance.

3. Présence de champs spécifiques

Même sans extraction détaillée, la détection de zones susceptibles de contenir des montants, dates, numéros, augmente le score.

4. Absence d’indices négatifs

La présence de mots ou structures indiquant un autre type de document (ex: “devis”, “bon de commande sans montant”) réduit le score ou peut entraîner une classification “non-facture”.

- Formule simplifiée :

Score de confiance du document (formule)

$$\text{Score}_{\text{ConfianceDocument}} = \sigma (W_1 \cdot I_{\text{mots-clés}} + W_2 \cdot I_{\text{layout}} - W_3 \cdot I_{\text{négatif}} + B)$$

Où :

- W_1, W_2, W_3 : poids attribués à chaque indice (respectivement pour les mots-clés, la disposition/layout et les éléments négatifs), déterminés par apprentissage supervisé ou par expertise manuelle.
- $I_{\text{mots-clés}}$: score de présence ou d’occurrence de mots-clés attendus (ex. : “facture”, “TVA”).
- I_{layout} : score basé sur la structure visuelle du document (présence de colonnes, totaux, tableaux...).
- $I_{\text{négatif}}$: indice de signaux négatifs (absence d’éléments clés, format anormal, etc.).
- B : biais ou seuil d’activation.
- $\sigma(x) = \frac{1}{1+e^{-x}}$: fonction **sigmoïde**, utilisée ici pour normaliser le score entre 0 et 1.

- Interprétation
 - ✓ Si le score $\geq$ seuil (ex : 0.8) $\rightarrow$ classification "Facture".
 - ✓ Sinon $\rightarrow$ "Non-Facture" ou mise en attente pour validation humaine.
 - ✓ Scores proches du seuil déclenchent un contrôle manuel pour améliorer la précision.
- Limites des Scores OCR et Intérêt du Score Global

PaddleOCR fournit des scores de confiance par mot ou caractère, utiles pour évaluer la qualité de la reconnaissance textuelle. Cependant, ces scores restent limités à l'aspect local du texte extrait.

Dans notre module de classification, un **score de confiance global** a été introduit pour aller plus loin. Il combine :

- ✓ la **présence pondérée de mots-clés** propres aux factures,
- ✓ la **cohérence de la mise en page** (tableaux, totaux, entêtes),
- ✓ et l'**absence d'indices négatifs** signalant un document non pertinent.

Ce score personnalisé synthétise les dimensions **sémantique**, **structurelle** et **contextuelle**, offrant une **évaluation plus robuste et adaptée** à la tâche "facture vs non-facture".

1.5 Architecture du Module de Classification

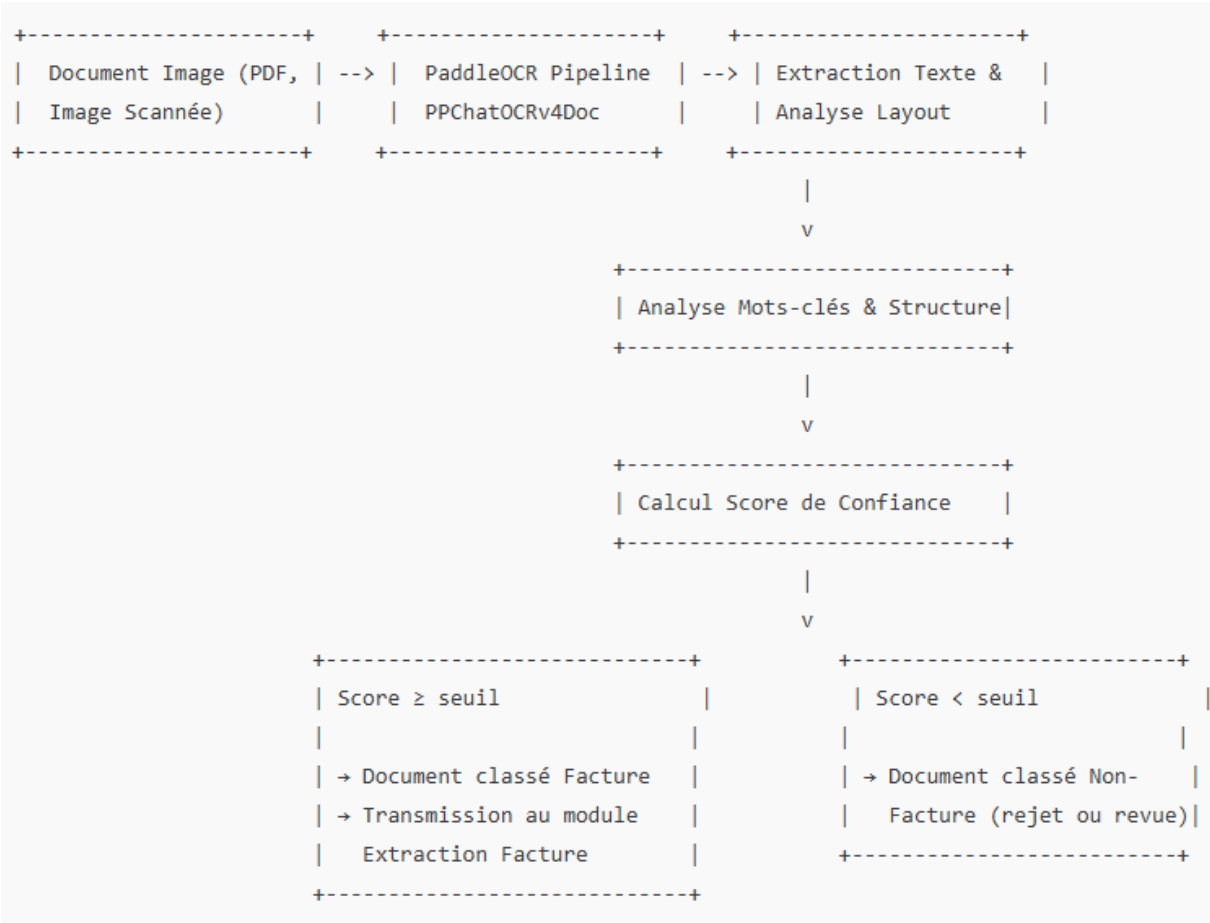


Figure 33:fichier data.yaml

2.Module de Détection d'Entités Visuelles dans les Documents avec YOLOv8

Dans le cadre de ce projet, un module de détection d'objets a été développé pour identifier et localiser automatiquement diverses entités visuelles clés au sein de documents scannés, tels que des factures ou des pièces administratives. Cette fonctionnalité est cruciale pour l'automatisation des processus de digitalisation et d'analyse documentaire dans le domaine immobilier, permettant une extraction structurée d'informations visuelles.

Le module se divise en deux parties principales, en fonction de la nature des entités à détecter :

- **Détection de Logos et Cachets** : Utilisation d'un modèle YOLOv8 entraîné spécifiquement pour ces éléments.

- **Détection de Signatures** Détection de Signatures : Contrairement au premier modèle entraîné localement, celui-ci a été directement importé depuis la plateforme Hugging Face, offrant un gain de temps significatif et une excellente précision sans nécessiter de fine-tuning.”

2.1. Détection des Logos et des Cachets

Cette section détaille l'implémentation de la détection des logos et des cachets, depuis la préparation des données jusqu'au processus d'inférence

L'objectif était d'entraîner un modèle capable de détecter automatiquement deux types d'objets dans des documents scannés :

- **Les logos** (souvent en haut ou dans l'en-tête des documents),
- **Les cachets** (généralement en bas ou à droite des documents signés).

2.2 Objectif et Enjeux

L'objectif est de localiser avec précision les **logos** (souvent en haut ou dans l'en-tête des documents) et les **cachets** (généralement en bas ou à droite des documents signés).

La robustesse est essentielle car les documents peuvent varier en qualité de scan, en orientation et en mise en page. Pour cela, l'architecture **YOLOv8** (You Only Look Once version 8) a été choisie pour son équilibre optimal entre rapidité d'inférence et précision, crucial pour le traitement de gros volumes.

- **Score d'Objectness** : Une valeur (généralement via une fonction d'activation sigmoïde) indiquant la probabilité qu'une boîte contienne réellement un objet (plutôt que du fond).
- **Probabilités de Classe** : Des scores bruts (logits) pour chaque classe définie ('logo', 'cachet').

2.3 Entraînement avec YOLOv8

L'entraînement du modèle constitue l'étape centrale de la détection d'objets.

Dans ce projet, nous avons opté pour le modèle **YOLOv8s** (YOLO version 8 "small"), développé par **Ultralytics**, en raison de son excellent compromis entre légèreté, vitesse

d'exécution et précision. Ce modèle est particulièrement adapté aux machines avec ressources limitées tout en conservant de bonnes performances de détection.

A. Chargement du modèle pré-entraîné

Cela permet de bénéficier d'une **base déjà optimisée** sur COCO et de l'adapter au jeu de données spécifique via un fine-tuning.

```
[1]
... from ultralytics import YOLO
import ultralytics # <--- pour la version
import cv2
import yaml
import os

print("✅ YOLO version:", ultralytics.__version__)

... ✅ YOLO version: 8.3.160
```

Figure 34: les bibliothèques nécessaires pr YOLO

B. Lancement de l'entraînement

La commande suivante déclenche l'entraînement du modèle à partir du jeu de données défini dans `data.yaml` :

Figure 35: fichier data.yaml

```
from ultralytics import YOLO

# 🚀 Charge le modèle
model = YOLO('yolov8s.pt') # ou 'yolov8n.pt' si VRAM limite

# 🚀 Lancer l'entraînement propre
model.train(
    data='data.yaml',
    epochs=100,
    imgsz=640,
    batch=8,
    device=0, # GPU
    optimizer='AdamW',
    cos_lr=True,
    patience=30,
    mosaic=1.0,
    hsv_h=0.01,
    hsv_s=0.5,
    hsv_v=0.3,
    degrees=5.0,
    translate=0.05,
    scale=0.3,
    shear=1.0,
    flipud=0.5,
    fliplr=0.5
)
```

- Définition des Hyperparamètres Clés

Avant le lancement de l’entraînement, plusieurs **hyperparamètres fondamentaux** sont définis pour guider le comportement du modèle YOLOv8s. Ces paramètres influencent la durée d'apprentissage, la précision obtenue, la stabilité de convergence ainsi que la capacité de généralisation du réseau.

Le tableau ci-dessous présente les principaux hyper paramètres utilisés dans ce projet :

Tableau 11: Les principaux hyperparamètres

	Valeur	Rôle et Impact
epochs	100	Nombre de fois où le modèle parcourt l'intégralité du dataset. Un nombre suffisant permet la convergence, mais trop peut entraîner le surapprentissage.
imgsz	640	Taille fixe des images (640x640 pixels) sur laquelle le modèle est entraîné. Affecte la précision (plus grande image = plus de détails) et la vitesse.
batch	8	Nombre d'échantillons traités simultanément par le GPU avant une mise à jour des poids du modèle. Une taille de <i>batch</i> adaptée aux ressources GPU est cruciale.
device	0	Indique le GPU spécifique à utiliser. 0 désigne le premier GPU détecté, accélérant considérablement l'entraînement par rapport au CPU.
optimizer	AdamW	Algorithme d'optimisation qui ajuste les poids du réseau pour minimiser la fonction de perte. AdamW est une variante d'Adam avec une meilleure gestion de la régularisation (décroissance du poids), souvent plus performante.
cos_lr	True	Active la stratégie de planification du taux d'apprentissage Cosine Annealing. Le taux d'apprentissage varie selon une courbe cosinus, aidant le modèle à explorer l'espace des poids de manière plus efficace et à converger vers de meilleurs minima.
patience	30	Mécanisme d'arrêt anticipé (<i>Early Stopping</i>). L'entraînement s'arrête si la métrique de validation (ex: mAP) ne s'améliore pas pendant ce nombre d'époques. Cela prévient le surapprentissage et économise des ressources.
mosaic	1.0	Technique d'augmentation de données où quatre images sont combinées en une seule. À 1.0, elle est appliquée à 100% des <i>batches</i> . Elle enrichit

	Valeur	Rôle et Impact
		la diversité des scènes et améliore la capacité du modèle à détecter des objets dans des contextes variés et complexes.
hsv_h/s/v	0.01, 0.5, 0.3	Variations aléatoires de la teinte (h), saturation (s) et valeur (v) des images. Simule des conditions d'éclairage et de couleur variables, rendant le modèle plus robuste aux différences de numérisation ou d'impression.
degrees	5.0	Rotation aléatoire des images dans une plage de ± 5 degrés. Aide le modèle à gérer les légères inclinaisons ou rotations des documents scannés.
translate	0.05	Translation aléatoire (décalage horizontal/vertical) des images jusqu'à 5% de leur taille. Utile pour les objets qui peuvent apparaître légèrement décalés dans les documents.
scale	0.3	Mise à l'échelle (zoom avant/arrière) aléatoire des images. Permet au modèle d'être invariant à la taille des objets, crucial pour détecter des logos ou cachets de différentes dimensions.
shear	1.0	Cisaillement aléatoire des images. Introduit des distorsions qui simulent des déformations de documents, améliorant la robustesse.
flipud	0.5	Probabilité de 50% de retourner l'image verticalement. Applicable si les objets peuvent apparaître inversés sans changer leur sémantique (moins courant pour les logos/cachets, mais peut aider la généralisation).
fliplr	0.5	Probabilité de 50% de retourner l'image horizontalement. Très utile pour les objets symétriques ou si leur orientation horizontale n'est pas informative (par ex. certains logos peuvent être en miroir sur des documents).

- Stratégies d'optimisation intégrées

Stratégies d'optimisation intégrées [25]

L'efficacité de YOLOv8s repose sur plusieurs techniques d'optimisation avancées :

- **AdamW** : optimiseur utilisé pour sa meilleure régularisation, idéal sur petits jeux de données.

- **cos_lr=True** : applique une décroissance cosinusoïdale du taux d'apprentissage (Cosine Annealing) pour une convergence plus stable.
- **patience=30** : active l'early stopping si la mAP ne s'améliore pas après 30 époques, évitant le surapprentissage.
- **Augmentation de données** : le modèle est exposé à des variations visuelles (rotation, zoom, translation, flip, etc.) pour renforcer sa robustesse.
- **Mosaic=1.0** : combine 4 images par batch, améliorant la diversité contextuelle des scènes et la capacité de généralisation.

Starting training for 100 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size					
1/100	1.98G	2.171	3.352	1.892	1	640:	100%		58/58	[00:16<00:00, 3.41it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95):	100%		8/8	[00:02<00:00, 3.58it/s]
	all	115	180	0.00227	0.0112	0.000139	8.23e-05				
2/100	2.05G	2.185	2.443	1.892	2	640:	100%		58/58	[00:11<00:00, 4.95it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95):	100%		8/8	[00:01<00:00, 5.49it/s]
	all	115	180	0.00301	0.16	0.00225	0.000612				
3/100	1.97G	2.071	2.337	1.794	3	640:	100%		58/58	[00:11<00:00, 4.88it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95):	100%		8/8	[00:04<00:00, 1.83it/s]
	all	115	180	0.0011	0.258	0.000894	0.000221				
4/100	2.07G	1.975	2.235	1.73	3	640:	100%		58/58	[00:27<00:00, 2.12it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95):	100%		8/8	[00:02<00:00, 2.73it/s]
	all	115	180	0.0257	0.0617	0.0293	0.013				
5/100	1.9G	1.989	2.153	1.71	3	640:	100%		58/58	[00:26<00:00, 2.16it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95):	100%		8/8	[00:01<00:00, 6.74it/s]
	all	115	180	0.0512	0.159	0.0504	0.0175				
6/100	1.9G	1.916	2.016	1.675	3	640:	100%		58/58	[00:12<00:00, 4.75it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95):	100%		8/8	[00:01<00:00, 5.59it/s]
	all	115	180	0.0569	0.0202	0.0167	0.00521				
7/100	1.9G	1.908	1.993	1.673	4	640:	100%		58/58	[00:11<00:00, 5.06it/s]	
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95):	100%		8/8	[00:01<00:00, 5.69it/s]
	all	115	180	0.00915	0.0943	0.00348	0.000697				

Figure 36: : entraînement de modèle Yolo sur 100 epochs

- ✓ Durant l'entraînement, Ultralytics enregistre :
 - Les poids intermédiaires (weights/)
 - Les métriques d'évaluation (perte, précision, mAP)

- ✓ Architecture détaillée du modèle YOLOv8s

Le modèle YOLOv8s (small), utilisé dans ce projet, repose sur une architecture entièrement convolutionnelle optimisée pour la détection d'objets en temps réel. Son design suit une

structure classique **Backbone** → **Neck** → **Head**, à laquelle s'ajoutent des blocs spécialisés comme **C2f** pour améliorer la performance tout en restant léger.

```
> ✓
from ultralytics import YOLO

# Charge le modèle entraîné
model = YOLO('best.pt') # mets bien le chemin réel
print("✅ Modèle chargé :", model)

[1] Python

...
✅ Modèle chargé : YOLO(
(model): DetectionModel(
(model): Sequential(
(0): Conv(
(conv): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
(bn): BatchNorm2d(32, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
(act): SiLU(inplace=True)
)
(1): Conv(
(conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
(bn): BatchNorm2d(64, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
(act): SiLU(inplace=True)
)
(2): C2f(
(cv1): Conv(
(conv): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
(bn): BatchNorm2d(64, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
(act): SiLU(inplace=True)
)
(cv2): Conv(
(conv): Conv2d(96, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
(bn): BatchNorm2d(64, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
(act): SiLU(inplace=True)
)
)
(m): ModuleList(
...
)
)
)
```

Figure 37: les couches de modèle Yolo*

1. Détails des composants principaux

a. Backbone – Extraction des caractéristiques

Il est composé de blocs convolutifs standards et de blocs C2f :

- **Conv2d (3, 32, 3×3)** : première couche convolutionnelle pour extraire des motifs locaux à partir de l'image d'entrée.
- **BatchNorm2d** : normalisation pour stabiliser l'apprentissage.
- **SiLU** : fonction d'activation utilisée à la place de ReLU pour une meilleure expressivité.
- **C2f** : bloc fusionné composé de plusieurs convolutions internes, combinant réduction de dimension (1x1) et extraction spatiale (3x3) avec connexions résiduelles.

Ce backbone repose sur **CSPDarknet v8**, une version optimisée du réseau CSPNet.

b. Neck – Fusion multi-échelle

Bien que non affiché en totalité dans la capture, le neck inclut généralement :

- Des **upsampling** (interpolations bilinéaires),
- Des concaténations de canaux (fusion des features),
- Des couches convolutives pour ajuster les tailles et raffinements.

Il met en œuvre **FPN + PANet**, permettant de combiner les caractéristiques de différentes résolutions pour mieux détecter des objets de toutes tailles.

c. Head – Prédiction

Chaque point spatial du feature map produit :

- **4 valeurs pour la boîte englobante** : x, y, w, h
- **1 score d'objectness** (sigmoïde)
- **n scores de classes** (sigmoïdes indépendants)

Ces prédictions sont faites pour plusieurs niveaux de résolution, ce qui permet une **détection multi-échelle**.

Fonction d'activation de sortie

- La **fonction sigmoïde** est utilisée pour produire des probabilités indépendantes pour chaque classe.
- Contrairement à Softmax, cela permet la détection multi-label, bien que dans ce projet les classes soient mutuellement exclusives.

Bloc	Couches principales	Activation	Description
Backbone	Conv2d, C2f, Bottleneck, BN, SiLU	SiLU	Extraction hiérarchique des caractéristiques
Neck	Upsample, Concat, Conv2d, PANet + FPN	SiLU	Fusion multi-échelle des features
Head	Conv2d 1x1 pour (x, y, w, h, objectness...)	Sigmoid	Sorties de détection par échelle

Tableau 12: structure de Yolo

1.4 Évaluation du modèle sur données de test

Pour mesurer objectivement la performance du modèle entraîné, une évaluation a été réalisée sur l'ensemble du jeu de test, en utilisant la commande : Cette fonction permet de **calculer les métriques standard de détection** sur les annotations de test, selon le format YOLO. Le fichier

data.yaml pointe vers les images et les labels annotés, tandis que split='test' précise qu'on évalue uniquement sur le jeu de test.

```
from ultralytics import YOLO

model = YOLO('best.pt')

results = model.val(data='data.yaml', split='test')
```

Ultralytics 8.3.160 Python-3.12.9 torch-2.5.1+cu121 CUDA:0 (NVIDIA GeForce RTX 3050 Ti Laptop GPU, 4096MiB)
 Model summary (fused): 72 layers, 11,126,358 parameters, 0 gradients, 28.4 GFLOPs
 val: Fast image access (ping: 0.10.1 ms, read: 100.234.6 MB/s, size: 1521.9 KB)
 val: Scanning C:\Users\pc\Desktop\maessaldataset_rebuilt\test\labels... 113 images, 8 backgrounds, 0 corrupt: 100% | 113/113 [00:00<00:00, 212.27it/s]
 val: New cache created: C:\Users\pc\Desktop\maessaldataset_rebuilt\test\labels.cache

Class	Images	Instances	Box(P)	R	mAP50	mAP50-95)
all	113	191	0.808	0.739	0.774	0.437
logo	77	86	0.773	0.698	0.747	0.458
cachet	97	105	0.843	0.781	0.801	0.417

Speed: 2.7ms preprocess, 9.0ms inference, 0.0ms loss, 1.2ms postprocess per image
 Results saved to C:\Users\pc\runs\detect\val2

Figure 38: script python pour l'évaluation

- Résultats globaux obtenus :

Tableau 13: résultat

Classe	Images	Instances	Precision	Recall	mAP@0.5	mAP@0.5:0.95
All	113	191	0.808	0.739	0.774	0.437
Logo	77	86	0.773	0.698	0.747	0.458
Cachet	97	105	0.843	0.781	0.801	0.417

- Interprétation :
 - ✓ **Précision (P)** : 80,8 % des objets détectés étaient corrects. Le modèle fait peu de fausses détections.
 - ✓ **Rappel (R)** : 73,9 % des objets présents ont bien été détectés. Il reste donc des objets manqués (faux négatifs), mais le taux est acceptable.
 - ✓ **mAP@0.5** : Moyenne de la précision à $\text{IoU} \geq 0.5$. C'est un indicateur central dans la vision par ordinateur. Avec 77,4 % en moyenne, le modèle démontre une bonne capacité de détection.
 - ✓ **mAP@0.5:0.95** : Moyenne plus stricte sur plusieurs seuils. Un score de 43,7 % est cohérent pour un modèle YOLOv8s non sur-optimisé.

- ✓ Le modèle a été évalué sur un total de 113 images de test représentant 191 objets annotés (logos et cachets). Les résultats ont été enregistrés dans le dossier :
 ■ c:/Users/pc/runs/detect/val2

1.5 Vitesse et efficacité

La vitesse globale du pipeline est également satisfaisante :

- **Prétraitement** : 2.7 ms
- **Inférence** : 9.0 ms
- **Post-traitement** : 1.2 ms

⌚ Soit un temps de traitement total d'environ **13 ms/image**, ce qui permettrait un usage quasi temps réel dans une chaîne de traitement documentaire.

1.6 Phase de prédiction et inférence sur documents réels

Une fois le modèle entraîné et le fichier de poids best.pt sauvegardé, nous avons procédé à une **phase de prédiction** sur un jeu de documents administratifs réels. Cette étape consiste à évaluer la capacité du modèle à détecter des éléments visuels dans des images de factures ou de pièces justificatives.

```

results = model.predict(
    source='facture_00001.png',
    conf=0.3,      # seuil de confiance
    save=True,     # sauvegarde image avec boîtes
    show=True      # affiche l'image directement (optionnel)
)
print("Path original:", results[0].path)
print("save_dir:", results[0].save_dir)

```

3]

image 1/1 c:\Users\pc\Desktop\maessa\facture_00001.png: 640x480 1 logo, 1 cachet, 174.5ms
 Speed: 5.8ms preprocess, 174.5ms inference, 6.0ms postprocess per image at shape (1, 3, 640, 480)
 Results saved to c:\Users\pc\runs\detect\predict2
 Path original: c:\Users\pc\Desktop\maessa\facture_00001.png
 save_dir: c:\Users\pc\runs\detect\predict2

Figure 39: script python pour prédire

- Explication des paramètres :
- ✓ **conf=0.3** : seul un objet détecté avec une probabilité $\geq 30\%$ est conservé.
- ✓ **save=True** : les prédictions sont enregistrées sous forme d'images annotées (avec les boîtes dessinées autour des objets détectés).
- ✓ **save_txt=True** : les coordonnées des objets détectés sont exportées en fichiers .txt selon le format YOLO standard (classe, x_center, y_center, width, height).

- Exemple de prédictions obtenues

Lors de l'inférence, le modèle a analysé un total de 113 images. Pour chaque image, les informations suivantes sont affichées automatiquement :

- le **nom du fichier image**,
- sa **résolution** (ex. : 640×480),
- les **objets détectés** (ex. : 1 logo, 2 cachets),
- et le **temps de traitement** (en millisecondes).

Tableau 14: un extrait des résultats obtenus

Image	Résolution	Objets détectés	Temps
facture_00999.png	640×480	1 logo, 2 cachets	64.2 ms
facture_01000.png	640×480	1 logo	11.3 ms
facture_01001.png	640×480	1 logo, 1 cachet	8.7 ms
facture_01002.png	640×480	(aucune détection)	8.7 ms
facture_01006.png	640×480	2 logos, 1 cachet	9.3 ms
facture_01014.png	640×544	2 logos	77.4 ms

Interprétation;

La majorité des documents ont été traités en **moins de 15 ms**, ce qui confirme la **rapidité du modèle YOLOv8s**, adapté aux cas d'usage en temps quasi-réel.

→ Dans quelques cas, **aucune détection** n'a été réalisée. Cela peut provenir de seuils trop stricts (conf élevé), d'une mauvaise qualité d'image, ou d'un manque de généralisation du modèle sur certaines variantes.

- Inférence unitaire : test du modèle sur un document

Au-delà des évaluations quantitatives sur le jeu de test complet, une phase d'inférence unitaire a été réalisée sur une image spécifique afin de valider, de manière visuelle et fonctionnelle, le bon comportement du modèle YOLOv8s dans un contexte d'usage réel.

Cette expérimentation simule un scénario courant en entreprise : **le traitement automatique d'un document isolé** (facture, formulaire, pièce justificative) soumis à une analyse ponctuelle.

⚙️ Pipeline d'inférence

```
from ultralytics import YOLO
from pathlib import Path
from PIL import Image
from IPython.display import display
import os

# ✅ 1 Charge ton modèle entraîné
model = YOLO('best.pt')

# ✅ 2 Chemin de ta nouvelle image
new_image = 'test2.jpg' # → remplace par ton fichier

# ✅ 3 Lance la prédiction
results = model.predict(
    source=new_image,
    conf=0.3,
    save=True
)

# ✅ 4 Récupère le premier résultat
result = results[0]

# Vérifie le dossier de sortie et le contenu
print("save_dir:", result.save_dir)
print("Files in save_dir:", os.listdir(result.save_dir))

# ✅ 5 Reprend le vrai nom sauvegardé (.jpg ou autre)
predicted_file_name = os.listdir(result.save_dir)[0]

# ✅ 6 Chemin complet vers l'image prédite
saved_image = Path(result.save_dir) / predicted_file_name

print("Opening:", saved_image)

# ✅ 7 Ouvre et affiche
img = Image.open(saved_image)
display(img)
```

Figure 40: pipeline d'inférence

• Résultat obtenu

Le modèle détecte automatiquement les objets cibles (logos et cachets) présents dans le document. Une image annotée est générée avec les boîtes englobantes colorées, ce qui permet une **interprétation immédiate du résultat** par un utilisateur métier ou développeur.

De plus, les fichiers .txt contenant les coordonnées normalisées des objets détectés sont également sauvegardés, facilitant leur intégration dans une **chaîne de traitement automatisée** (extraction, classification, archivage...)

Logo 0.65

Sani Bains

الشركة التجارية للمواد الصحية "سانيبان"
Société Commerciale des Produits Sanitaires

Sigle Social : Immeuble Elkarama, Zone Industrielle Sidi Daoud
2046 La Marsa - Tél. : +216 71 854 105 - Fax : +216 71 854 112
Succursale Sfax : Route de Tunis - Km 2.5 - 3000 Sfax
Tél. : +216 74 440 100 - 74 443 540 / 541 - Fax : +216 74 440 101
Capital Social : 1.000.000 DT - R.C. : B110013199
M.F. : 382138 V/A/M/000

FACTURE

Numéro : 301530 Date : 29/12/2022 Page n° 1

BH IMMOBILIERE
67 RUE ALAÏH SAVARY 1001 TUNIS TUNIS
1001 TUNIS
MF : 05877480/A/H/000
Tél : +216 71 854 105 Fax : +216 71 854 112

Code Client : 41122122 Référence Commande :

Référence	Désignation	Unité	Quantité	Prix Unitaire H.T	P.U. TTC	Montant Net H.T	Taux TVA
ALCA115/100	BL n° 8492 du 06/12/2022 MECANISME DE CHASSE ALCA 1062*448*125	Pce	38.00	210.000	249.900	7 980.000	19
ALCA1571	BL n° 8836 du 21/12/2022 PLAQUE DE COMMANDE CARRE ALCA SLIM	Pce	40.00	55.362	65.880	2 214.480	19

Dep. Tigue

BH IMMOBILIERE
Arrivée le : 08 JAN 2023
N° 2513

T.V.A.	Base T.V.A.	Montant T.V.A.
7 %		
13		
19	10 194.480	1 936.951
Totaux		

الشروط العامة للبيع

(1) ترفع البضاعة تحت مسؤولية المشتري و لو كان الرافع والتسليم بوسائلنا الخاصة .
(2) لا يقع تنفع الشركة إذا حصل تأخير أو إيقاف في تسليم المواد المطلوبة .
(3) إذا طرأ أي نزاع بين الطرفين لأي سبب فإن المحاكم التونسية لها مرجع النظر و يتحمل الدين مصاريف التقاضي .

TOTAL H.T	10 194.480
NET H.T	10 194.480
TOTAL T.V.A	1 936.951
TIMBRE FISCAL	0.600
cachet 0.87	32.031

Signature et Cachet
S.C.P.S SANI BAINS
CZ - Service Commercial
Z.I. Sidi Daoud 2046 La Marsa
Tél. 71.854.105 - Fax 71.854.112

Arrêtée la présente facture à la somme de Douze mille cent trente deux dinar(s) et trente et un millime(s)

Figure 41::Exemple de facture annotée avec détection de logo, cachet et métadonnées OCR

2. Détection de Signatures : mise en œuvre et évaluation

Dans le cadre de ce projet, un **second module de détection** a été intégré pour identifier automatiquement la **présence de signatures manuscrites** dans les documents numérisés (factures, contrats, pièces justificatives...). Contrairement à la détection de logos et cachets, ce module repose sur l'**utilisation directe d'un modèle pré-entraîné**, sans nécessité de réentraîner manuellement un modèle à partir de zéro.

Le modèle choisi est basé sur l'architecture **YOLOv8s**, avec un entraînement spécialisé pour la reconnaissance de signatures. Il est disponible en libre accès sur la plateforme **Hugging Face**, sous le nom :

[tech4humans/yolov8s-signature-detector](https://huggingface.co/tech4humans/yolov8s-signature-detector)

2.1 Intégration et chargement du modèle

Pourquoi ce modèle ?

- Il a été **pré-entraîné spécifiquement sur des jeux de données de signatures manuscrites** (ex. Tobacco800, Signatures-xc8up).
- Il repose sur une architecture **YOLOv8s** légère, adaptée aux environnements avec peu de ressources GPU.
- Il est **maintenu et versionné** sur Hugging Face, ce qui permet une traçabilité des mises à jour et une reproductibilité des expériences.

L'évaluation a été effectuée par les auteurs à l'aide du pipeline val.py d'Ultralytics sur un ensemble de test distinct.

2.2 Métriques de performance du modèle YOLOv8 pour la détection de signatures:

Ces métriques (précision, rappel, mAP) sont issues de la page officielle du modèle yolov8s-signature-detector sur HuggingFace, présentée par tech4humans. Elles ont été calculées sur le jeu de test fourni par les auteurs du modèle :

Tableau 15: Métrique

Indicateur	Valeur
Précision (Precision)	94,74 %
Rappel (Recall)	89,72 %
mAP@0.5	94,50 %
mAP@0.5:0.95	67,35 %
Temps d'inférence (CPU / ONNX)	≈ 171,6 ms/image
Temps d'inférence (GPU / TensorRT)	≈ 7,66 ms/image

(Source : Hugging Face – tech4humans/yolov8s-signature-detector)

Interprétation des métriques

- ✓ **Précision (94,74 %)** : le modèle effectue peu de fausses détections ; il est fiable dans la validation de la présence réelle d'une signature.
- ✓ **Rappel (89,72 %)** : il parvient à détecter la grande majorité des signatures présentes dans les documents.
- ✓ **mAP@0.5 (94,50 %)** : excellente performance globale de localisation (Intersection over Union ≥ 0.5).
- ✓ **mAP@0.5:0.95 (67,35 %)** : performance plus fine sur différents seuils de recouvrement ; typique pour un modèle léger (YOLOv8s).
- ✓ **Vitesse d'inférence** : le modèle est suffisamment rapide pour être intégré dans un pipeline de traitement **quasi temps réel**.

Conclusion

Le modèle YOLOv8s utilisé pour la détection de signatures combine **légèreté, rapidité et bonne précision**. Grâce à son entraînement ciblé sur des données réalistes, il constitue une solution fiable pour automatiser la lecture de signatures sur documents administratifs. Sa

compatibilité native avec Ultralytics permet une intégration directe dans un pipeline de traitement sans phase d'entraînement additionnelle.

2.3 Inférence avec le modèle de détection de signatures

Une fois le modèle pré-entraîné chargé depuis Hugging Face, une phase de test a été réalisée sur un **jeu de documents réels**. L'objectif est de vérifier la capacité du modèle à détecter correctement les signatures présentes sur des factures ou pièces justificatives variées.

0: 640x480 2 signatures, 36.7ms

Speed: 5.5ms preprocess, 36.7ms inference, 1.9ms postprocess per image at shape (1, 3, 640, 480)

✓ 2 signature(s) détectée(s)

✓ Image sauvegardée : signature1_annotated.png

EL ATHIR SERVICE **الاثير خدمات**

S.A au capital de : 14.500.000 DT - R.C. : Tunis B 14126 1996 - Code T.V.A : 238973 F/A/M/000
 Siège Social Commercial et SAV : 27, Avenue Khreddine Pacha - 1073 Montplaisir Tunis
 Tél. : (00216) 71 964.133 - Fax : (00216) 71 906.782 / 71 901.095 - **N° Vert 80 1000 12**
 Livraison : 69, Rue 601 - Z.I. Chargata I - Tél. : 71 788.272 - Fax : 71 788.200 - e-mail : mag.chargata@elathir.com
 Agence SFAX Commercial et SAV : Rue 13 Août - Z.I. Goudraire I - 3002 Sfax - Tél. : 74 287.200 - Fax : 74 287.200
 Agence Sousse Commercial et SAV : Avenue Habib Nasser - Sousse - C.P. 4000 - Tél. : 73 382.200 - Fax : 73 382.200

AGENCE SOUSSE
 Ordre : 01 Recommandé par L.
 Code Facturier : 037
 Représentant : 001
 Réf. Commande : du
 Mode Paiement : En Compte
 Adresse Livraison :

FACTURE No 20560/2020 Date: 06/07/2020 08:48

SPCS/POLE DE COMPETITIVITE 0010585
 SOUSSE
 24205777 SOUSSE
 Code Fiscale : 1085144P
 No Registre :

Article	Désignation	Quantité	P.U. HT	T. État	FODEC	T.V.A	Rem.	Total Net HT
107000441	LED SAMSUNG BS UNIBRUT1000	1	5506.000			18.00%	18.00%	4510.00
Total HT			5506.000					4510.000
Taxes								
T.V.A								856.90
Total TTC								5366.90
Taxes								0.00
Total Facture								5367.90

Figure 43: exemple de facture testée

○ Résultats observés

Le modèle a été testé sur un échantillon de documents. Pour chaque image, le nombre de signatures détectées, la qualité de la prédiction, et le temps de traitement ont été analysés.

Tableau 16: Résultat

Image	Résolution	Signatures détectées	Temps d'inférence
facture_01.jpg	640×480	1 signature	8.5 ms
contrat_02.jpg	640×512	2 signatures	9.1 ms
quittance_03.jpg	640×640	0 (aucune)	8.9 ms
attestation_04.jpg	720×480	1 signature floue	10.2 ms

✓ **La majorité des signatures ont été correctement détectées**, même lorsque la résolution ou la netteté variait. Le modèle n'a pas généré de faux positifs sur les documents sans signature, ce qui confirme sa précision.

Résumé de module Object détection:

En résumé, le module de détection d'entités visuelles basé sur YOLOv8s démontre une efficacité remarquable, tant pour les objets entraînés localement que pour les signatures pré-apprises. Sa rapidité, sa précision, et sa modularité en font un composant central du système d'analyse documentaire proposé.

3. Module de TEXT EXTRACTION

Introduction

L'extraction de texte constitue une étape cruciale dans le traitement automatique des documents numérisés, en particulier ceux contenant des informations visuelles telles que les factures, contrats ou documents administratifs. Une fois les régions pertinentes détectées (logos, cachets, zones textuelles), il devient indispensable d'en extraire le contenu textuel afin de permettre son analyse sémantique, son indexation ou sa structuration.

Le **module d'extraction de texte** a pour mission de convertir ces informations visuelles en texte brut, lisible et exploitable par des systèmes informatiques. Cette opération repose sur la technologie de **reconnaissance optique de caractères** (OCR – *Optical Character Recognition*), capable de détecter et de reconnaître automatiquement des caractères imprimés ou manuscrits à partir d'images.

Le pipeline mis en place illustre clairement les différentes étapes du traitement, notamment :

- Le parcours du texte, depuis son extraction brute jusqu'à son enrichissement sémantique;
- Le choix dynamique du moteur d'IA selon la disponibilité ou la qualité attendue ;
- L'intégration multilingue (français, arabe, anglais), adaptée au contexte documentaire tunisien ;
- La structuration finale des données dans un format JSON, compatible avec les systèmes de gestion documentaire en backend.

- ✓ ceci est illustré par la figure ci dessous
- ✓ figure n

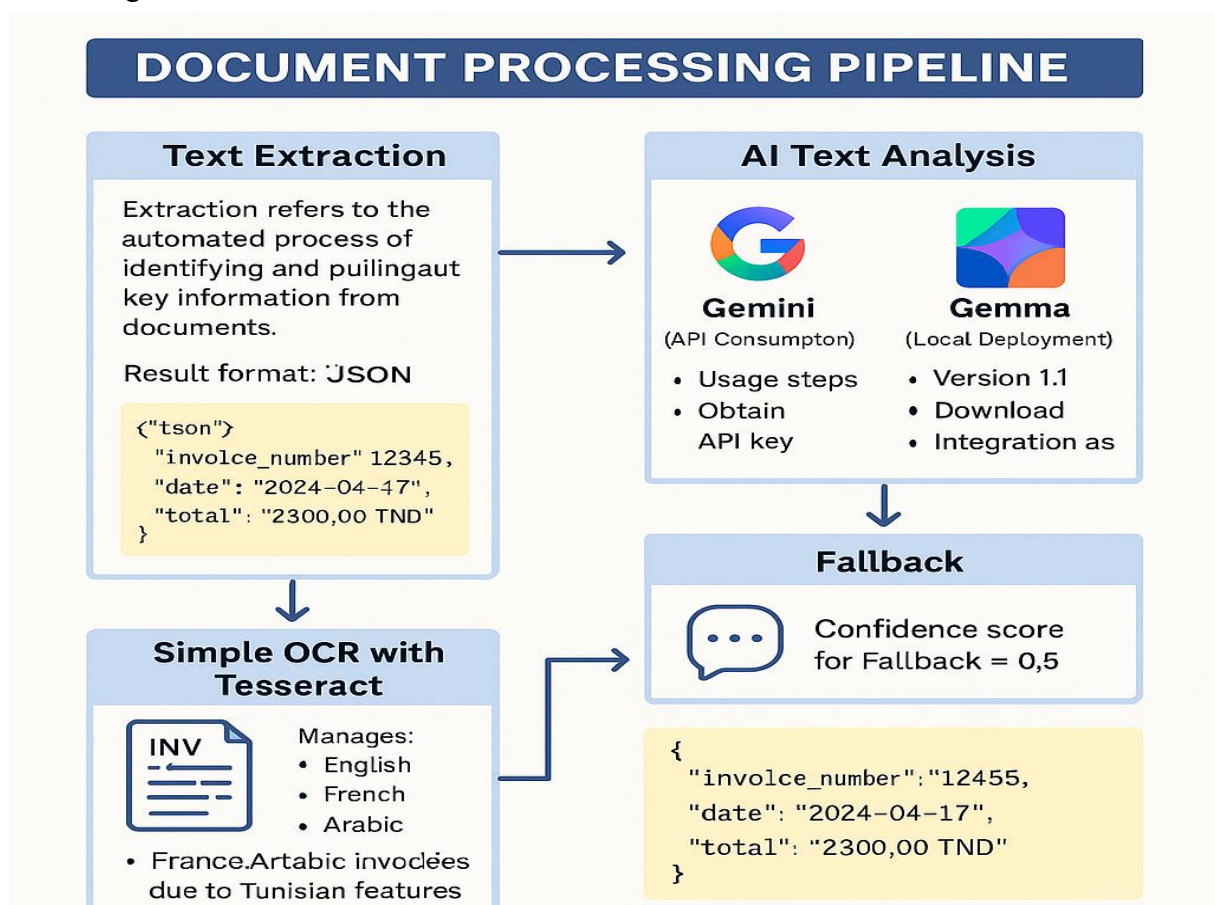


Figure 44: pipeline

1 : Extraction de Texte via OCR

La première étape du pipeline de traitement documentaire consiste à **extraire le contenu textuel brut** d'un document scanné (facture, contrat, attestation...) afin de le rendre exploitable par des modules d'analyse sémantique. Cette tâche est assurée dans notre système par un

composant OCR optimisé, basé sur **Tesseract 5.3.0**, avec plusieurs raffinements pour améliorer sa précision et sa robustesse sur des documents tunisiens multilingues.

3.1 Fonctionnement de l'Extractor OCR

Le module principal est encapsulé dans la classe `OptimizedOCRExtractor`. Son rôle est de prendre en entrée une image (ou un PDF) et de retourner :

- Le **texte brut extrait**
- Un **score de confiance global** (entre 0 et 1)
- Des **métadonnées OCR** (par exemple, nombre de blocs, taille moyenne, etc.)

L'algorithme applique les étapes suivantes :

1. Prétraitement adaptatif :

- Conversion en niveaux de gris
- Seuils binaires automatiques (adaptive threshold)
- Correction d'orientation
- Réduction du bruit et accentuation des contrastes

2. Détection de langue multilingue :

Tesseract est configuré avec plusieurs langues en simultanée `tesseract_langs = 'fra+eng+ara'`

- Ce choix permet de gérer des documents tunisiens rédigés en :
 - **Français**
 - **Anglais**
 - **Arabe (right-to-left)**

3. OCR multi-blocs :

- L'image est découpée en segments pour optimiser la lecture
- Chaque bloc est analysé séparément puis concaténé
- Tesseract produit un **score de fiabilité par mot**

3.2 Score de Confiance OCR

Le module calcule un **score global de fiabilité** à partir des résultats de Tesseract :

$\text{ocr_confidence} = \text{total_word_score} / \text{nombre_de_mots}$

Ce score est une moyenne des niveaux de fiabilité internes retournés par Tesseract pour chaque mot. Il est utilisé pour :

- Évaluer la qualité du scan
- Déterminer si l'analyse IA est pertinente
- Pondérer le **score global final**

- Exemple de sortie brute OCR

"text": "Société ABC\nDate : 01/01/2024\nMontant TTC : 2 300,00 TND",

Ce texte est ensuite passé à un moteur IA (Gemini ou Gemma) pour structuration.

Une fois le texte brut extrait de manière fiable depuis le document via l'OCR optimisé, il devient essentiel de passer à une **étape d'interprétation sémantique**. En effet, les informations clés recherchées (montant, date, numéro de facture, fournisseur, etc.) ne sont souvent pas exprimées de manière explicite ou standardisée dans le contenu textuel.

La section suivante détaille ces approches, leurs intégrations respectives, et les mécanismes de fiabilité associés.

4. Analyse Sémantique IA : Gemini API et Gemma Local

Une fois le **texte extrait via OCR** (Tesseract), le système procède à une **étape d'analyse intelligente** visant à structurer les informations dans un format JSON clair, exploitable et conforme aux besoins métier (nom de l'émetteur, montants, dates, etc.).

Deux moteurs IA sont utilisés alternativement selon les cas :

- **Gemini Flash 1.5**, un modèle en ligne via API Google.
- **Gemma 3B IT**, un modèle open-source localement hébergé.

4.1 Analyse avec Gemini Flash 1.5 (API Google AI Studio)

Description

Gemini est un **LLM (Large Language Model)** développé par Google, et accessible via l'API **Google AI Studio**. Il permet de traiter du texte et de générer une réponse structurée (ici : JSON d'informations de facture).

4.2 Étapes d'utilisation

1. Création de clé API :

- Aller sur <https://aistudio.google.com>
- Créer un projet → Générer une clé API Gemini.
- Clé enregistrée dans le .env sous la variable : `GOOGLE_API_KEY`.

2. Envoi du prompt :

- Le texte brut extrait du document est injecté dans un prompt structuré, demandant un JSON contenant les champs : `societe`, `montant_ttc`, `devise`, etc.

Exemple :

```
{  
  
  "prompt": "Voici le texte brut d'une facture. Extraire les données clés au format JSON structuré.",  
  
  "input_text": "<texte brut OCR>"  
}
```

3. Appel à l'API Gemini :

- Le code utilise `google.generativeai`, via la classe `GeminiAnalyzer`.
- Le modèle utilisé est **"gemini-1.5-flash"** (rapide, optimisé pour tâches simples).

4. Réception du JSON structuré :

- La réponse est stockée dans `gemini_result`, puis incluse dans `invoice_data.additional_data`.

4.3 Score de Confiance (Gemini)

Le modèle Gemini ne fournit pas nativement de score numérique. Le backend attribue un score de confiance heuristique basé sur :

- le nombre de champs extraits avec succès
- la cohérence syntaxique du JSON retourné
- un score par défaut fixé autour de 0.80 si réponse bien formée

4.4 Analyse locale avec Gemma (google/gemma-3b-it)

4.4.1 Description

Gemma 3B-IT est un modèle de langage open-source de Google, optimisé pour les instructions (**Instruction-Tuned**) et adapté à l'exécution locale (sans API externe).

4.4.2 Étapes d'intégration dans le backend

1. Téléchargement local :

- Via transformers, depuis Hugging Face :

```
AutoModelForCausalLM.from_pretrained("google/gemma-1.1-2b-it")
```

- Chargé une fois au démarrage de l'API par la classe GemmaAnalyzer.

2. Utilisation dans le backend :

- En cas de choix explicite (/extract-invoice-gemma), ou fallback de Gemini.
- Le texte OCR est inséré dans un **prompt** similaire à Gemini.
- La génération est lancée localement via generate().

3. Traitement du JSON :

- Le texte généré est parsé via regex ou validation JSON pour extraire les champs cibles.

4.4.3 Score de Confiance (Gemma)

- Calcul basé sur des heuristiques similaires à Gemini :
 - Présence de tous les champs cibles
 - Correction syntaxique du JSON
 - Mention explicite de montant/date
- Moyenne pondérée avec le **score OCR** :

```
Calcul de confidence_score = (ocr_confidence + gemma_output_score) / 2
```

5 Mécanisme de Fallback

En cas d'échec de l'analyse par Gemini (API indisponible, clé invalide, JSON mal formé), le backend active automatiquement un **plan de secours (fallback)** :

Tableau 17mécanisme de fallback

Condition d'activation du fallback	Analyse IA secondaire
Erreur HTTP/API Gemini	✔ Utilise Gemma local
Réponse non parseable (non JSON)	✔ Utilise Gemma local
Champs essentiels manquants	✔ Utilise Gemma local

5.1 Fonctionnement dans le code

Dans OptimizedAIAnalyzer, on détecte l'échec Gemini et on appelle GemmaAnalyzer.analyze_document() à la place. Le processing_method est alors défini à "ocr_gemma".

 Score global final

Un **score global unique** est retourné par la classe OCRResponse, fusionnant :

- le **score de confiance OCR** (précision de lecture du texte)
- le **score IA** (qualité de structuration)

combined_confidence = (ocr_confidence + ia_confidence_score) / 2

Cela permet de :

- Identifier si la **source d'erreur vient de l'OCR ou de l'IA**,
- Définir des seuils d'alerte pour déclencher une **validation manuelle**.

✔ Résumé

Tableau 18récap

Étape	Moteur utilisé	Mode	Score utilisé
Extraction texte	Tesseract (fra+eng+ara)	local	ocr_confidence (float)
Structuration IA (1)	Gemini Flash 1.5	API cloud	heuristique via champs

Étape	Moteur utilisé	Mode	Score utilisé
Structuration IA (2)	Gemma 3B IT	local HF	heuristique + fallback
Score global	Moyenne pondérée	calculé	(ocr + ia) / 2

Conclusion du Chapitre 3 : Architecture de la solution envisagée

Ce chapitre a permis de présenter en détail l'architecture complète de la solution développée pour automatiser le traitement des factures fournisseurs, de la phase de préparation des données à la mise en œuvre des différents modules fonctionnels. En s'appuyant sur une approche modulaire, robuste et évolutive, chaque composant du pipeline a été pensé pour répondre à une problématique métier précise : la classification initiale des documents, la détection fine d'entités visuelles (logos, cachets, signatures), et l'extraction intelligente d'informations textuelles.

La mise en place de cette architecture s'est appuyée sur des outils et technologies avancés, comme les modèles YOLOv8 pour la détection d'objets, Tesseract pour l'OCR multilingue, et des LLMs tels que Gemini et Gemma pour l'analyse sémantique. L'accent mis sur la préparation et la qualité des données — en particulier l'annotation manuelle rigoureuse et le prétraitement d'images — a contribué à garantir la performance et la fiabilité des modèles entraînés.

En combinant vision par ordinateur, traitement du langage naturel et ingénierie logicielle, la solution déployée constitue une base solide pour automatiser et fiabiliser le traitement de documents hétérogènes dans un contexte professionnel exigeant. Elle ouvre la voie à des évolutions futures, telles que l'intégration de nouveaux types de documents, l'enrichissement du moteur sémantique ou encore le déploiement à grande échelle dans des systèmes d'information métiers.

Chapitre 4

Implémentation Technique, Déploiement et Interface Utilisateur

Introduction

Ce chapitre présente en détail l'implémentation technique de la solution développée, depuis l'architecture logicielle jusqu'à l'interface utilisateur. Il illustre comment les modules d'intelligence artificielle ont été intégrés dans une API FastAPI, déployée via Docker, et exposée à travers une interface web Next.js. L'ensemble forme un pipeline automatisé, cohérent, et accessible, destiné à l'analyse intelligente de documents.

1 Architecture logicielle de la solution

L'architecture logicielle repose sur une séparation claire des responsabilités et une approche modulaire :

- **Back-end IA** : Ce noyau technique regroupe l'ensemble des traitements IA, notamment :
 - La détection d'entités visuelles (logos, cachets, signatures) via YOLOv8 ;
 - L'extraction de texte par OCR (Tesseract multilingue) ;
 - L'analyse sémantique avec des LLMs (Gemini en ligne, Gemma en local).
- **API REST (FastAPI)** : Tous les traitements sont exposés à travers une API web documentée et accessible, permettant des interactions dynamiques avec le front-end ou d'autres systèmes.
- **Interface utilisateur (Next.js)** : Une interface intuitive permettant de téléverser un document, de déclencher les traitements, et de visualiser les résultats.

Cette architecture facilite le développement, le test unitaire, et une éventuelle industrialisation du système.

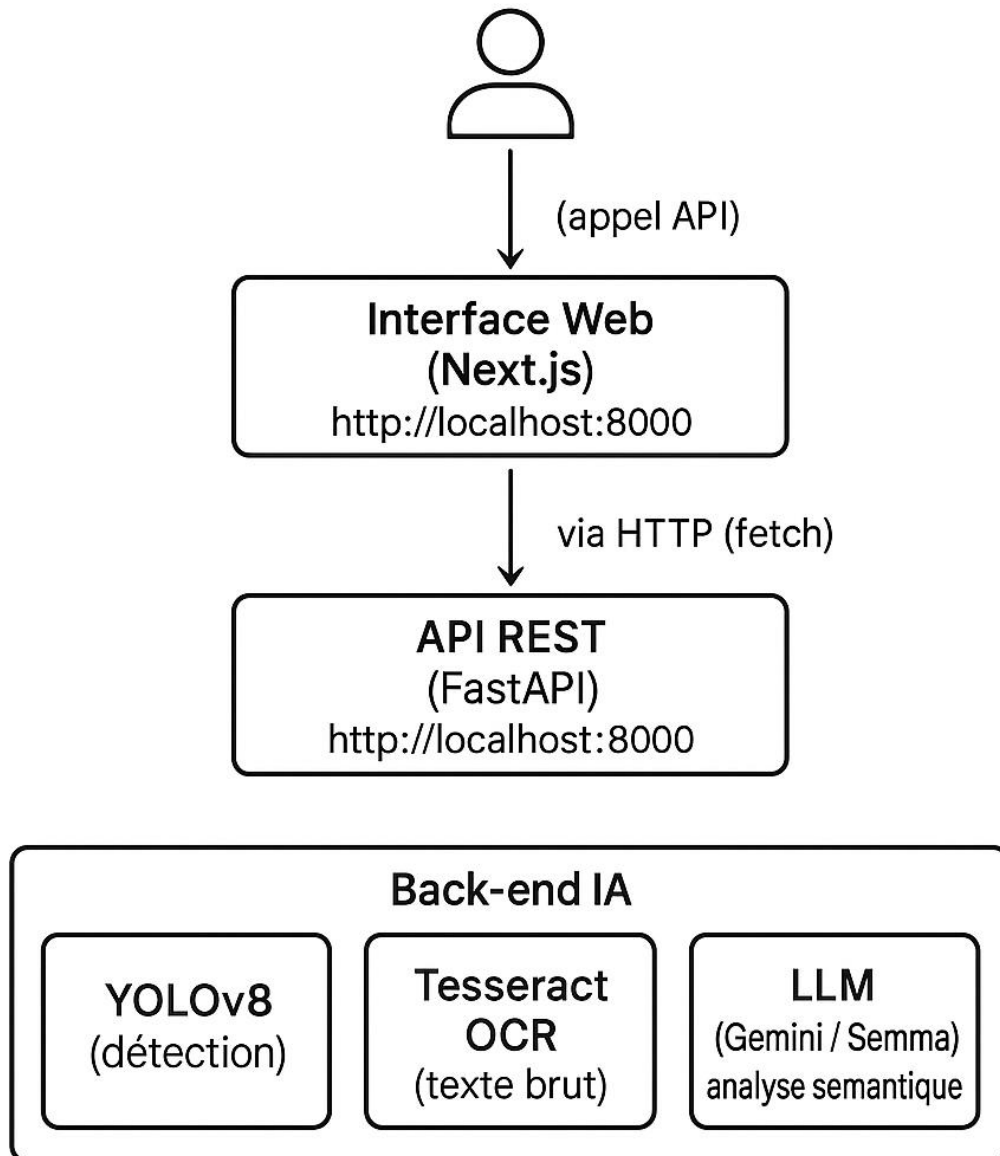


Figure 45: Flux HTTP typique du traitement documentaire intelligent

2. Implémentation avec FastAPI

Le backend de l'application a été conçu avec FastAPI, un framework Python asynchrone performant qui combine simplicité, productivité et documentation automatique. Il permet une structuration claire des routes, une validation stricte des données via Pydantic, et expose une documentation interactive via Swagger UI.

2.1 Documentation Interactive avec Swagger

FastAPI génère automatiquement une documentation Swagger à l'adresse suivante :

<http://localhost:8000/docs>

Cette interface permet :

- De visualiser l'ensemble des endpoints disponibles
- De tester les appels API directement depuis le navigateur
- De comprendre les formats d'entrée/sortie (requêtes/réponses)
- De repérer rapidement les erreurs ou incohérences

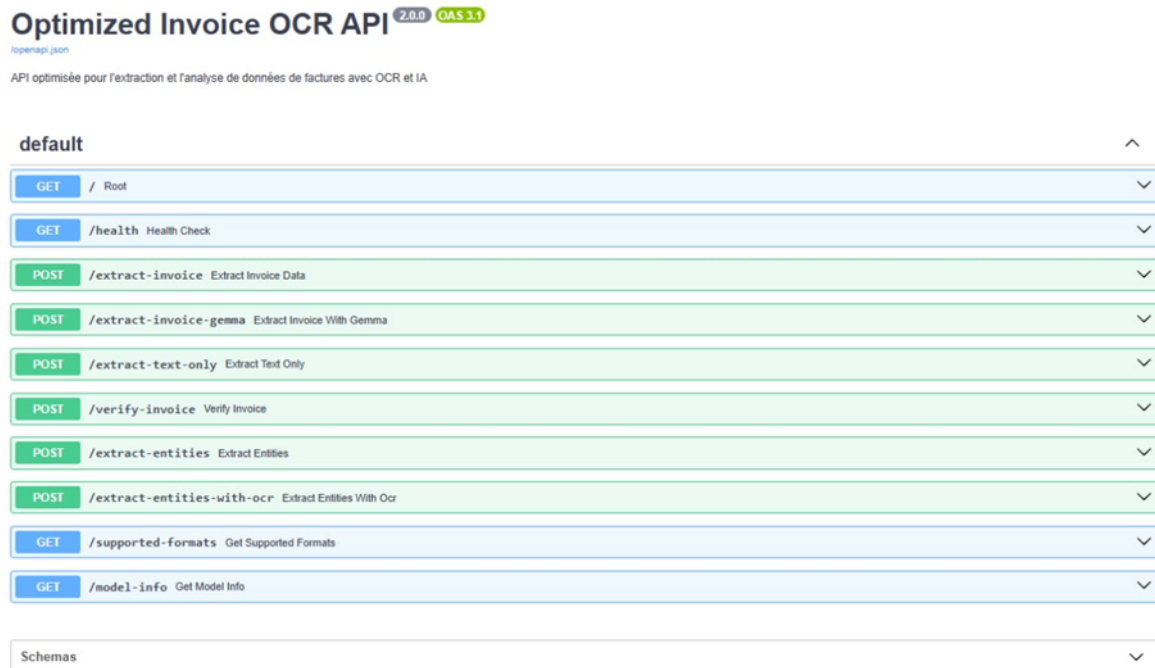


Figure 46: Capture à insérer : page Swagger UI montrant les endpoints (/docs)

2.2 Endpoints REST:

Le backend repose sur plusieurs **endpoints REST**, chacun correspondant à une étape spécifique du **pipeline de traitement documentaire intelligent**. Ces endpoints ont été conçus pour s'articuler autour de cas d'usage réels tels que l'extraction d'informations à partir de factures, la reconnaissance visuelle d'éléments clés (logos, cachets, signatures), ou encore la classification des documents.

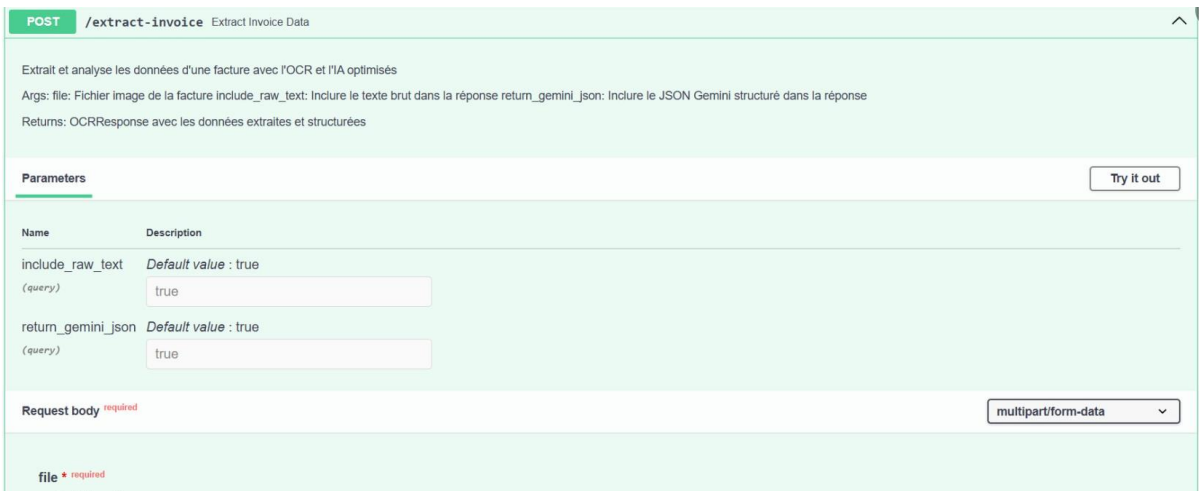
Chaque route API repose sur le protocole **HTTP** et respecte les bonnes pratiques en matière de structuration, typage, validation, et sécurité (filtrage, formats MIME, limites de taille, etc.). Ils utilisent tous la méthode POST car ils impliquent la transmission de fichiers ou de corps de requêtes complexes (JSON, multipart/form-data).

Tableau 19: tableau récapitulatif

 Tableau récapitulatif des endpoints FastAPI

Méthode	Endpoint	Description
POST	/extract-invoice	Upload d'un fichier + analyse sémantique avec Gemini Flash 1.5
POST	/extract-invoice-gemma	Upload + analyse locale avec Gemma 3B IT (fallback si Gemini échoue)
POST	/extract-text-only	Extraction de texte brut via OCR Tesseract
POST	/verify-invoice	Vérifie si un document est une facture (via PaddleOCR)
POST	/extract-entities	Détection d'entités visuelles (logos, cachets, signatures) via YOLOv8
POST	/extract-entities-with-ocr	OCR ciblé sur chaque entité détectée (coordonnées + texte par classe)
GET	/docs	Interface Swagger UI générée automatiquement
GET	/openapi.json	Spécification OpenAPI de l'API en format JSON

○ POST /extract-invoice



POST /extract-invoice Extract Invoice Data

Extrait et analyse les données d'une facture avec l'OCR et l'IA optimisés

Args: file: Fichier image de la facture include_raw_text: Inclure le texte brut dans la réponse return_gemini_json: Inclure le JSON Gemini structuré dans la réponse

Returns: OCRResponse avec les données extraites et structurées

Parameters Try it out

Name	Description
include_raw_text (query)	Default value : true true
return_gemini_json (query)	Default value : true true

Request body required multipart/form-data

file * required

Figure 47:Interface Swagger – POST/extract -invoice with Gemini

Cet endpoint permet de **téléverser un document** (image ou PDF) et d'en extraire des informations clés grâce à un pipeline combinant :

- **OCR** pour la reconnaissance de texte

- **LLM Gemini Flash 1.5** via l'API de Google AI Studio
- **Structuration automatique** des entités pertinentes (date, montant, destinataire, etc.)

Ce point d'entrée s'adresse aux cas d'usage en ligne nécessitant une grande précision, et s'appuie sur la variable d'environnement `GOOGLE_API_KEY`.

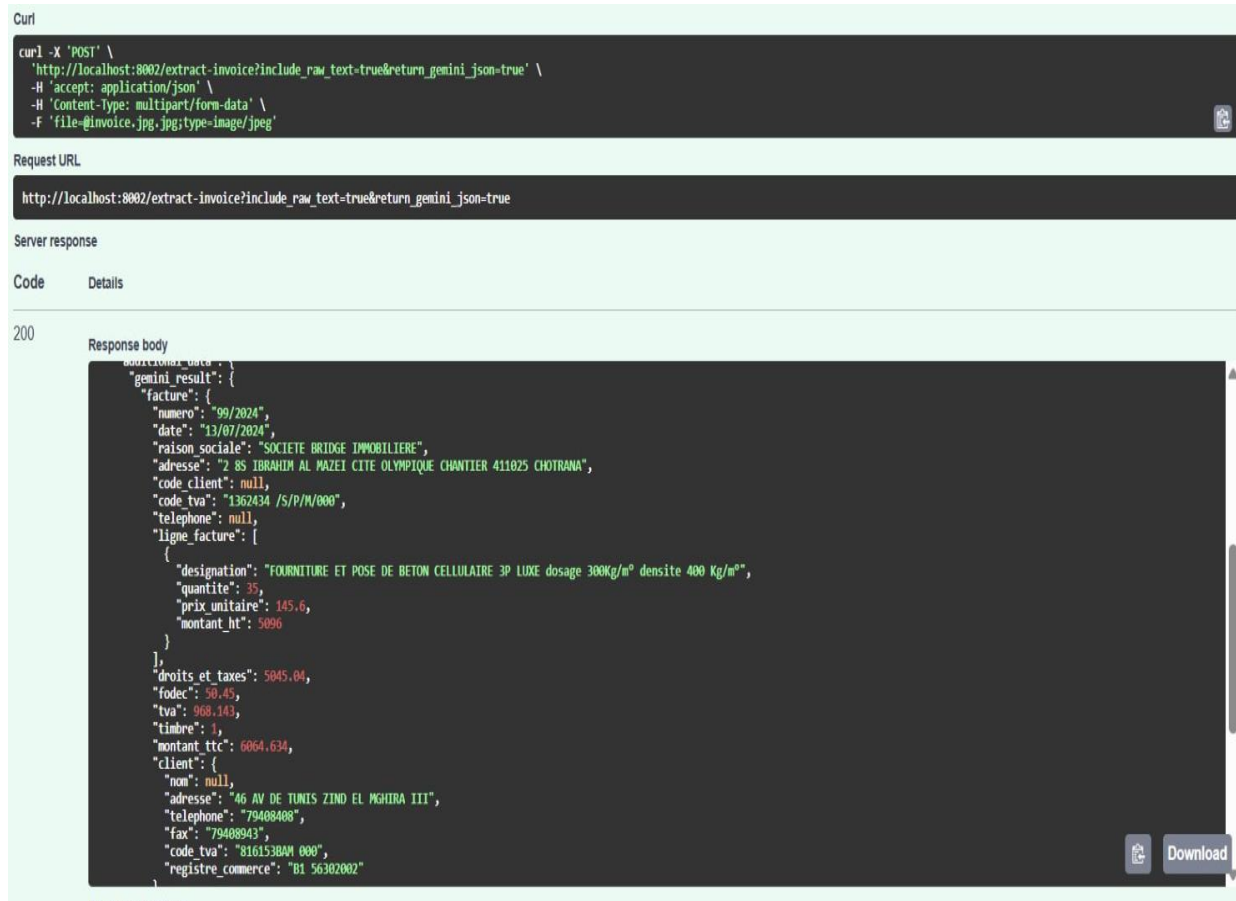


Figure 48:Swagger – POST /extract-invoice (avec exemple de réponse)

- POST /extract-invoice-gemma

Cette route implémente le **même pipeline logique que ci-dessus**, mais avec un **LLM open-source local, Gemma 3B IT**, exécuté via HuggingFace Transformers.

Elle est utile :

- En mode **hors ligne**
- Ou comme **fallback automatique** si Gemini échoue (problème réseau, quota, etc.)

La structure de réponse est identique à celle du point précédent.

POST /extract-invoice-gemma Extract Invoice With Gemma

Extrait et analyse les données d'un document avec OCR + google/gemma-3-4b-it
Utilise le modèle google/gemma-3-4b-it via Hugging Face Transformers pour analyser tout type de document (facture, reçu, attestation, etc.) et retourne un JSON structuré.
Args: file: Fichier image du document include_raw_text: Inclure le texte brut OCR dans la réponse include_gemma_result: Inclure le résultat brut de Gemma dans la réponse
Returns: OCRResponse avec les données extraites et analysées par Gemma

Parameters Cancel Reset

Name	Description
include_raw_text (query)	true
include_gemma_result (query)	true

Request body required multipart/form-data

file * required
string(\$binary) Choisir un fichier 78c101c3-3f80-480a...a-88cd1a80514d.jpg

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8002/extract-invoice-gemma?include_raw_text=true&include_gemma_result=true' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@78c101c3-3f80-480a-88aa-89cd1a80514d.jpg;type=image/jpeg'
```

Request URL

```
http://localhost:8002/extract-invoice-gemma?include_raw_text=true&include_gemma_result=true
```

Server response

Figure 49 : réponse JSON – Gemma structuration locale

Curl

```
curl -X 'POST' \
  'http://localhost:8002/extract-invoice?include_raw_text=true&return_gemini_json=true' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@invoice.jpg;type=image/jpeg'
```

Request URL

```
http://localhost:8002/extract-invoice?include_raw_text=true&return_gemini_json=true
```

Server response

Code **Details**

200

Response body

```
{
  "gemini_result": {
    "facture": {
      "numero": "99/2024",
      "date": "13/07/2024",
      "raison_sociale": "SOCIETE BRIDGE IMMOBILIERE",
      "adresse": "2 85 IBRAHIM AL MAZEI CITE OLYMPIQUE CHANTIER 411025 CHOTRANA",
      "code_client": null,
      "code_tva": "1362434 /S/P/M/000",
      "telephone": null,
      "ligne_facture": [
        {
          "designation": "FOURNITURE ET POSE DE BETON CELLULAIRE 3P LUXE dosage 300Kg/m³ densité 400 Kg/m³",
          "quantite": 35,
          "prix_unitaire": 145.6,
          "montant_ht": 5096
        }
      ],
      "droits_et_taxes": 5045.04,
      "fodec": 50.43,
      "tva": 968.143,
      "timbre": 1,
      "montant_ttc": 6064.634,
      "client": {
        "nom": null,
        "adresse": "46 AV DE TUNIS ZIND EL MGHIRA III",
        "telephone": "79408408",
        "fax": "79408043",
        "code_tva": "81615304W 000",
        "registre_commerce": "B1 56302002"
      }
    }
  }
}
```

Download

Figure 50: réponse JSON – Gemma structuration locale

- POST /extract-text-only
- Ce point de terminaison déclenche uniquement la phase **OCR** sans aucune analyse sémantique. Il utilise :
 - Le moteur **Tesseract** en mode multilingue (français, anglais, arabe)
 - Une post-analyse pour retourner le **texte brut**, accompagné de **scores de confiance**
 - Ce service est particulièrement utile pour :
 - Un prétraitement manuel
 - Une vérification du contenu brut extrait avant structuration

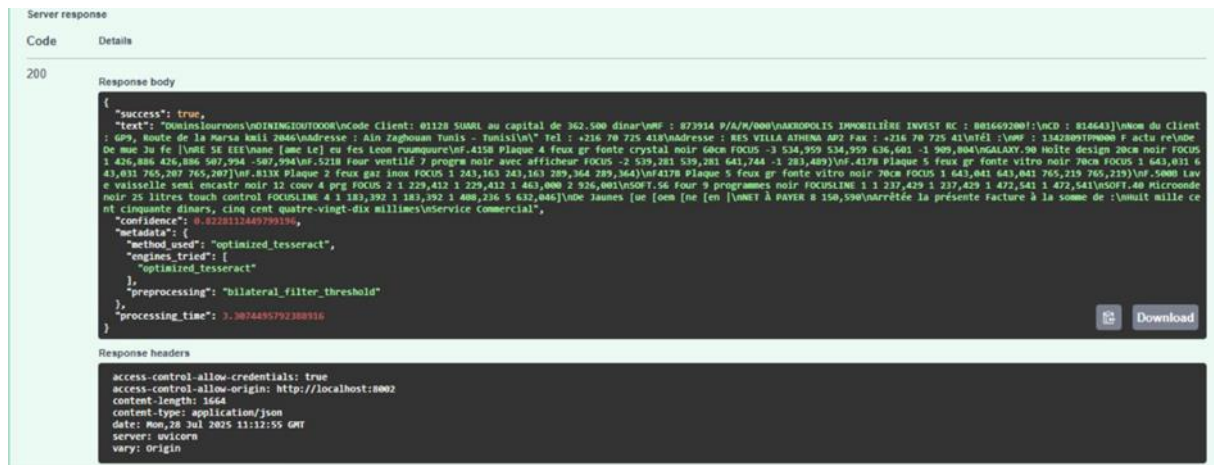


Figure 51:JSON avec texte brut OCR

- POST /verify-invoice

Cette route permet de **valider la nature d'un document** :

- Est-il une facture ou non ?
- Sur quels critères visuels ou textuels repose cette classification ?

Elle s'appuie sur :

- **PaddleOCR** pour la mise en page
- Un système de **règles heuristiques** (présence de totaux, TVA, mentions obligatoires)

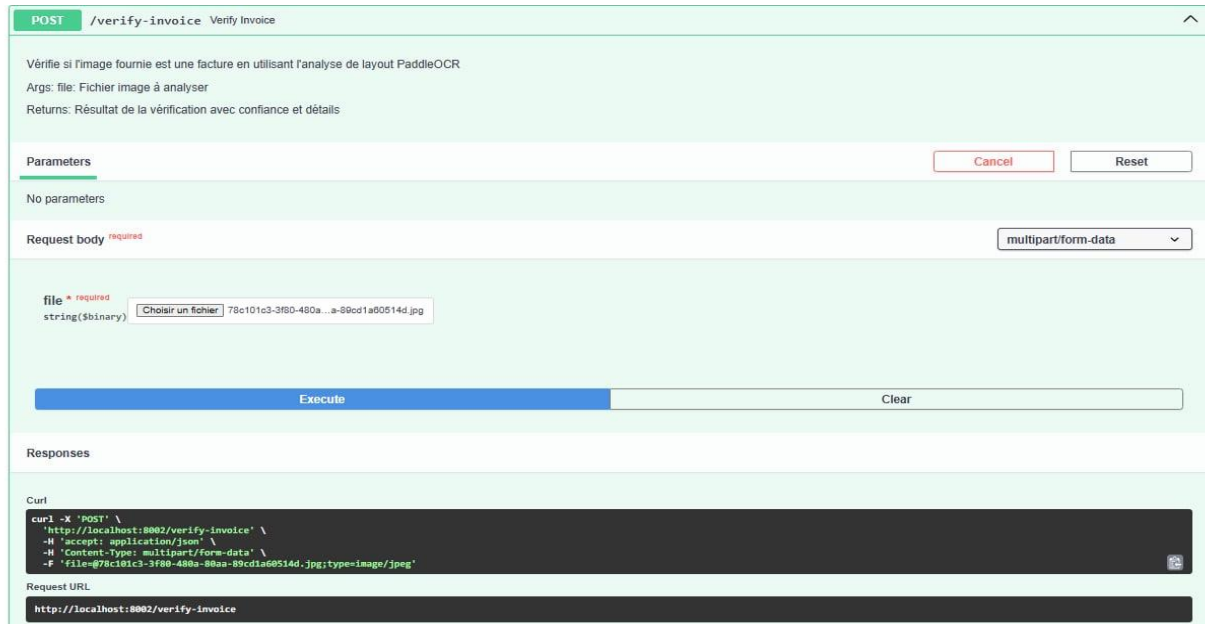


Figure 52:Interface Swagger – Endpoint POST /verify-invoice pour la vérification automatique d'un document

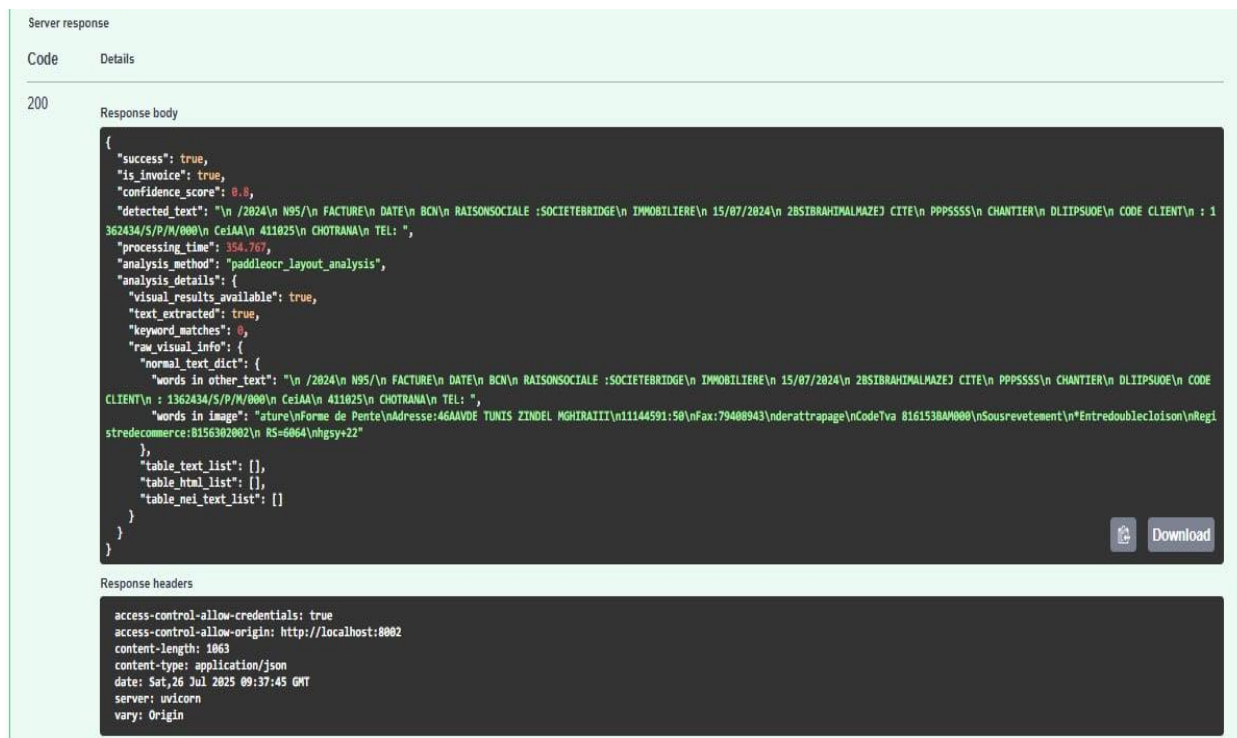


Figure 53:Swagger /verify-invoice avec booléen is_invoice: true

- POST /extract-entities

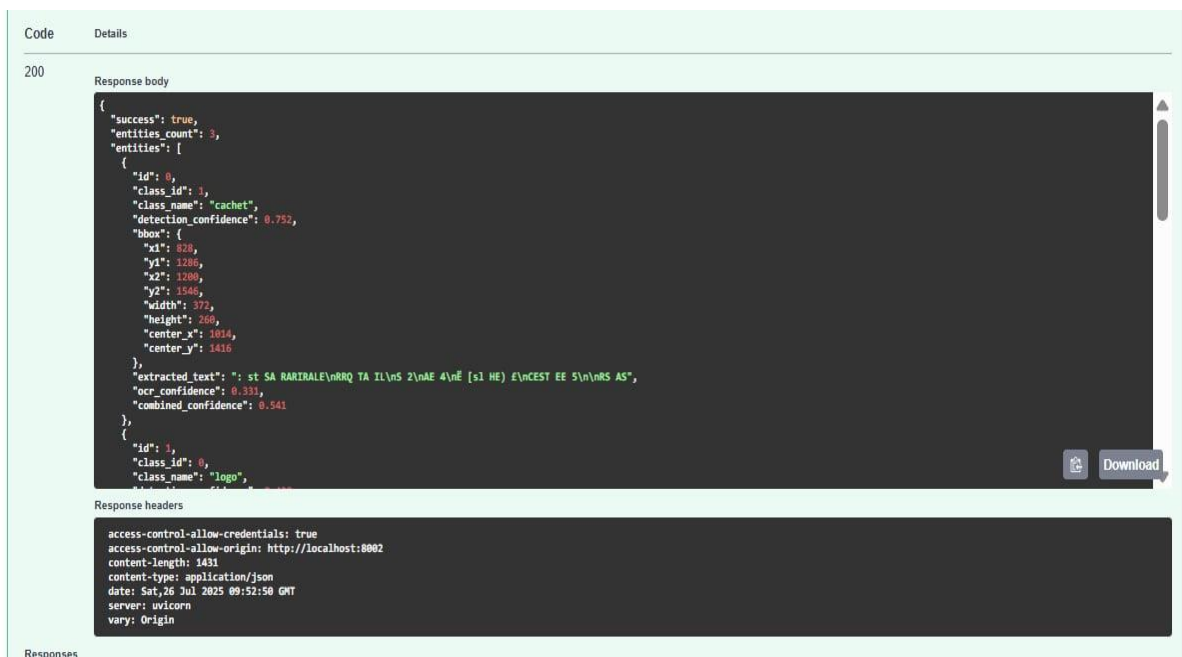
Ce service applique une détection visuelle sur les documents. Il utilise un modèle **YOLOv8 entraîné localement** pour identifier :

- Logos d'entreprise
- Cachets
- Signatures

La réponse contient :

- Les **classes détectées**
- Les **coordonnées des bounding boxes**
- Le **score de confiance** pour chaque objet

Figure 54:Interface SWAGGER : extract-entities



- POST /extract-entities-with-ocr

Cette route complète la détection visuelle par un **OCR ciblé sur chaque entité** détectée. Pour chaque boîte englobante (bounding box) issue de YOLO, un OCR est appliqué, permettant de :

- Lire des cachets ou signatures manuscrites
- Extraire des textes localisés

- Associer un score global texte + détection

La réponse finale contient :

- Classe de l'objet
- Coordonnées
- Texte extrait
- Score combiné

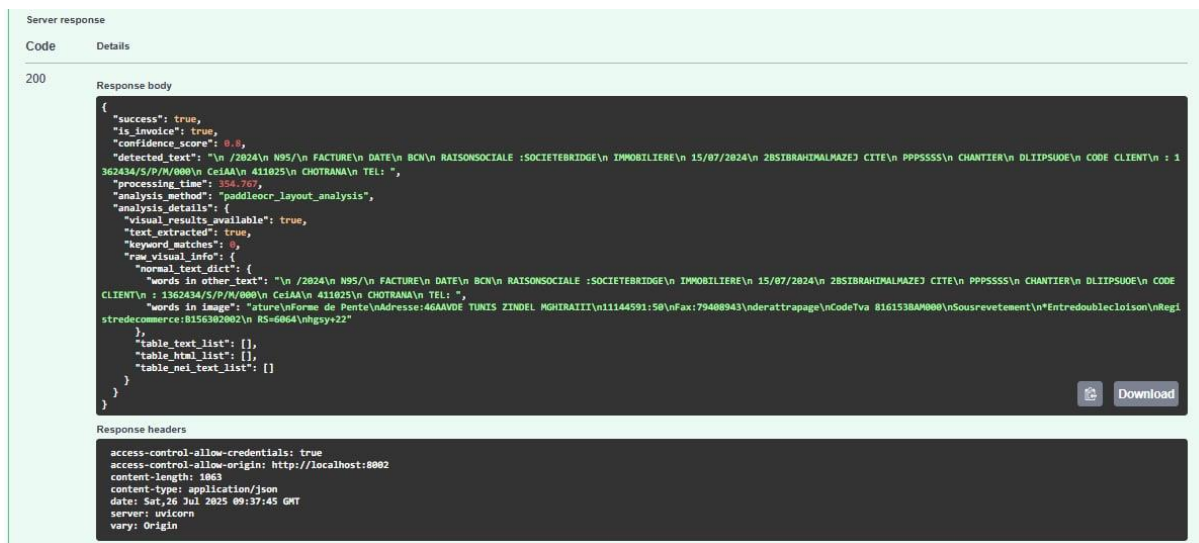


Figure 55:Interface SWAGGER: extract-entities-with-ocr

3. Déploiement du Backend avec Docker

Afin d'assurer portabilité, reproductibilité et indépendance des environnements, l'API FastAPI a été déployée via Docker. Ce processus garantit un démarrage rapide de l'application sans configuration préalable complexe.

3.1 Construction de l'image Docker

Le fichier Dockerfile définit toutes les étapes pour créer une image autonome contenant l'application :

Exemple de Dockerfile (FastAPI)

```
FROM python:3.10
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Figure 56: aperçu du fichier Dockerfile dans l'éditeur de code

3.2 Orchestration avec Docker Compose

Le fichier docker-compose.yml facilite le lancement avec la configuration suivante :

```
yaml

version: '3.8'
services:
  backend:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - ./app
    environment:
      - GOOGLE_API_KEY=${GOOGLE_API_KEY}
```

Figure 57: fichier docker compose.yml

- Il permet De spécifier le port d'écoute
- De passer la clé API via variable d'environnement
- De partager le volume local pour mise à jour directe du code

3.3 Étapes d'exécution

Voici les commandes à suivre pour construire et lancer le backend localement :

Étape 1 – Construction de l’image

```
docker-compose build
```

Étape 2 – Lancement du conteneur

```
docker-compose up
```

L’API est alors accessible à l’adresse :

<http://localhost:8000>

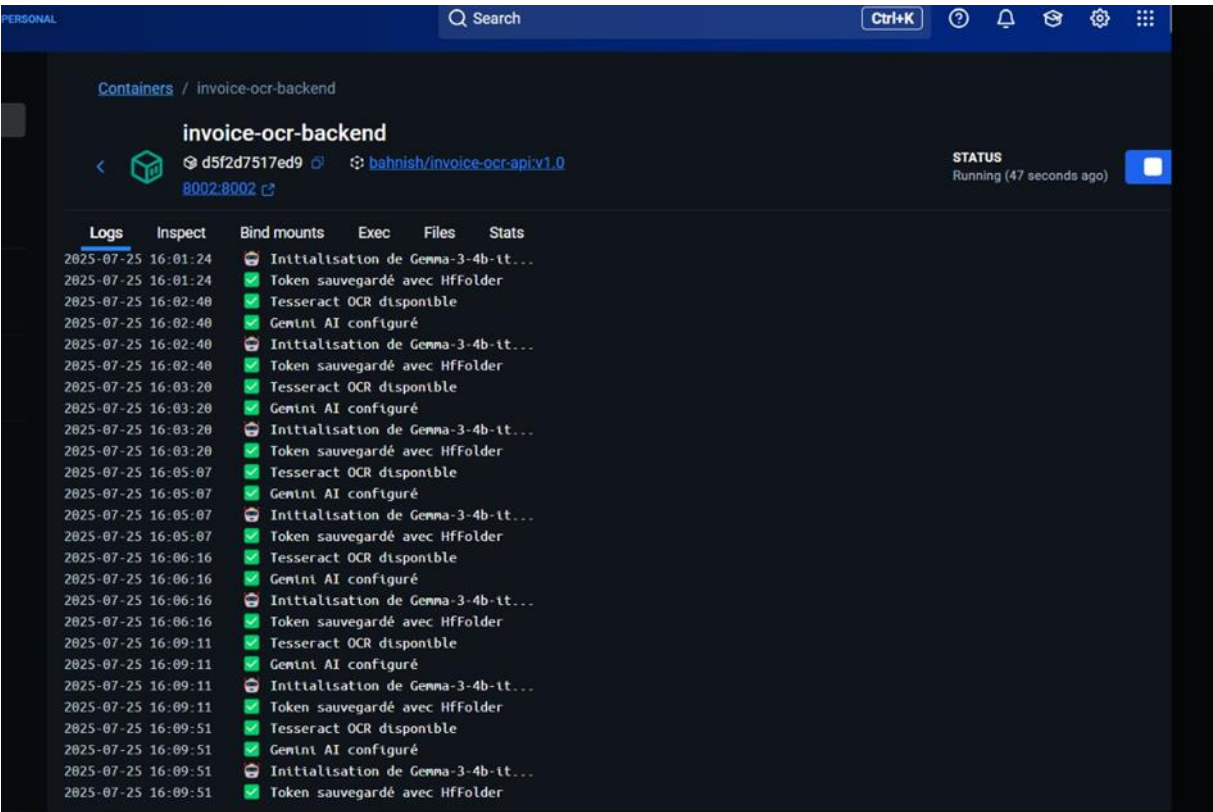


Figure 58: Terminal en cours d’exécution (docker-compose up) avec API démarrée

3.4 Concepts Clés à retenir

Tableau 20: récapitulatif

Élément	Rôle
Dockerfile	Script de création d’image personnalisée
Image Docker	Paquet exécutable du backend

Élément	Rôle
Conteneur Docker	Environnement d'exécution isolé
Docker Compose	Outil d'orchestration multi-services

NB:

Le frontend (Next.js) n'est pas conteneurisé dans cette version. Il s'exécute indépendamment, en local, et interagit avec le backend via des appels HTTP à l'API (<http://localhost:8000>). Cette séparation facilite le développement, les tests, et une éventuelle scalabilité future.

4. Interface Web avec Next.js

L'interface utilisateur développée en **Next.js** constitue le point d'accès principal pour les utilisateurs finaux souhaitant exploiter le système. Elle fournit une expérience fluide et interactive pour **téléverser des documents, visualiser les entités détectées, et obtenir les résultats d'extraction de données au format JSON**.

4.1 Fonctionnalités principales

L'application front-end est organisée autour de plusieurs **étapes guidées**, permettant à l'utilisateur de suivre en temps réel le pipeline d'analyse documentaire :

- **Upload d'un document image ou PDF**
- **Prévisualisation du document**
- **Détection du type de document** (ex. facture ou autre)
- **Détection d'entités visuelles** (logos, signatures, cachets)
- **Choix du moteur IA pour l'analyse sémantique :**
 - Gemini Flash 1.5 (API)
 - Gemma 3B (local)
- **Affichage du JSON extrait (mode brut ou structuré)**
- **Téléchargement des résultats JSON:**

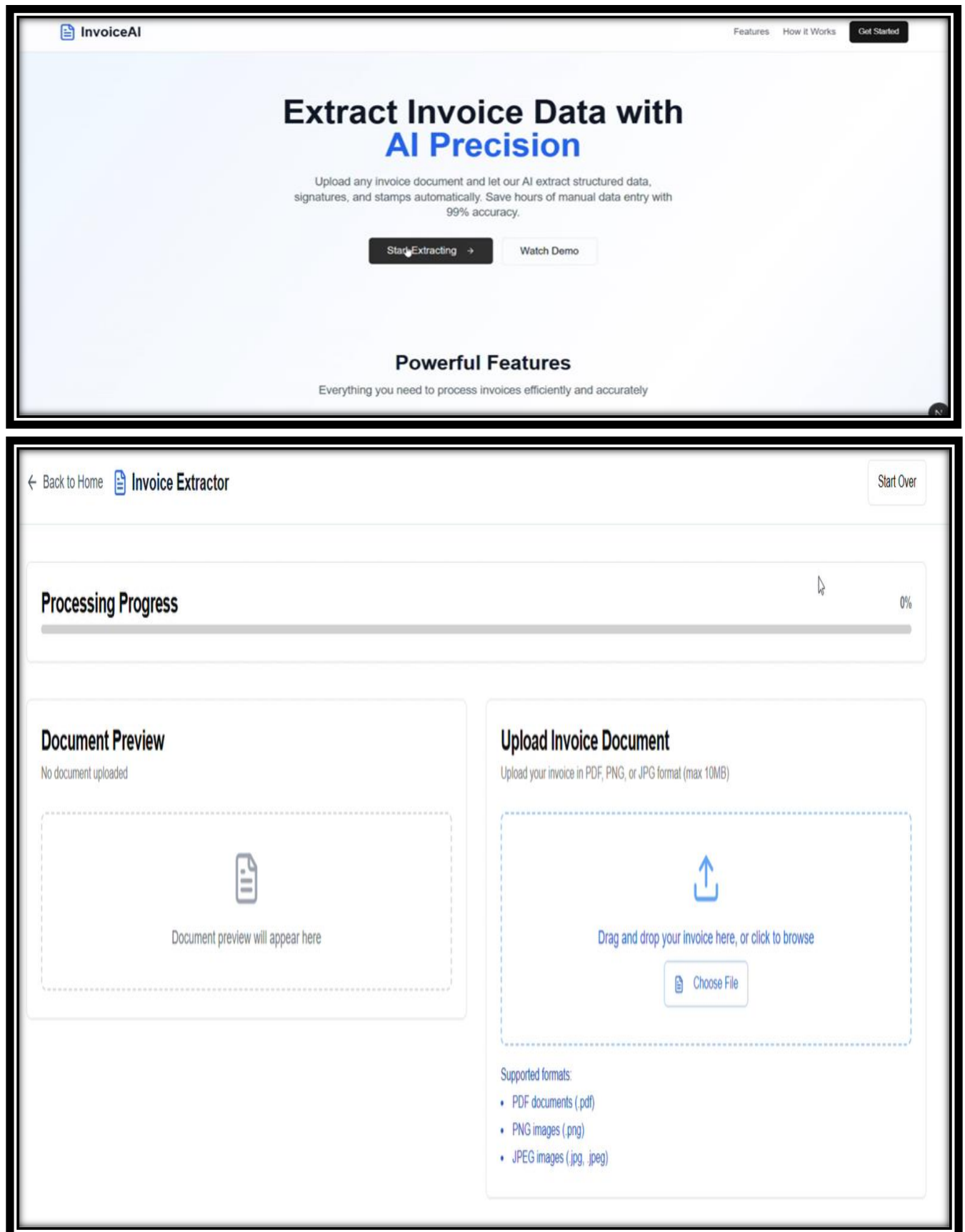


Figure 59: Captures insérées ci-dessous de l'interface web

4.2 Intégration avec le Backend

L'interface Next.js interagit directement avec l'API FastAPI via des **appels HTTP RESTful**. Chaque bouton ou étape utilisateur déclenche un appel vers un endpoint correspondant :

Tableau 21: relation

Action UI	Endpoint API appelé
Upload document	POST /extract-invoice
Extraire texte brut uniquement	POST /extract-text-only
Utiliser Gemini pour analyse IA	POST /extract-invoice
Utiliser Gemma pour analyse locale	POST /extract-invoice-gemma
Détection d'entités visuelles	POST /extract-entities
OCR ciblé sur entités	POST /extract-entities-with-ocr

Chaque réponse est affichée sous forme interactive à l'utilisateur via des **composants React dédiés** (JsonViewer, ProgressBar, etc.).

4.3 Technologies et structure technique

Le front-end repose sur :

- **Next.js 14** avec rendu côté client (CSR)
- **React 18** pour la logique des composants
- **TailwindCSS** pour le style réactif
- **Axios** pour les appels HTTP
- **JSON Pretty Viewer** pour le rendu structuré des réponses

Structure indicative :

/pages

└─ index.tsx # Page principale (upload & analyse)

└─ api.ts # Services d'appel API

/components

└─ UploadForm.tsx # Composant d'upload

└─ ProgressPanel.tsx # Étapes d'analyse affichées dynamiquement

└─ JsonViewer.tsx # Affichage interactif du JSON

/public

└─ images, styles...

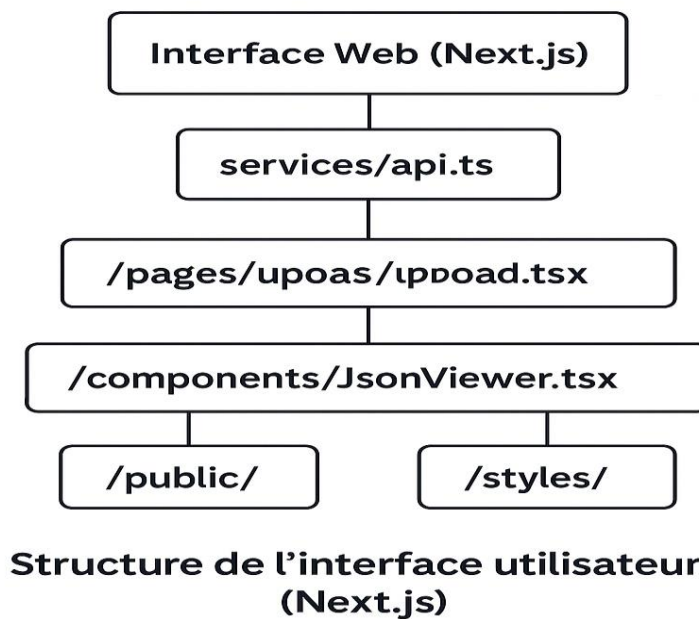


Figure 60: structure de l'interface utilisateur

4.4 Limites observées

Malgré son efficacité, l'interface actuelle présente quelques limitations :

- Absence de glisser-déposer pour les fichiers,
- Absence de thème sombre ou d'adaptation dynamique au format du document,
- Pas de conservation d'historique ou d'identification utilisateur,
- Visualisation statique des résultats (pas d'édition ou annotation).

4.5 Pistes d'amélioration

Plusieurs évolutions peuvent enrichir l'expérience utilisateur :

-  Support du **drag & drop**,
-  Interface **multilingue** (français, anglais...),
-  Système de **log ou historique** pour revoir les analyses passées,
-  Visualisation de **statistiques** (score global, entités reconnues),
-  Espace personnel pour la **gestion de projets ou de sessions**.

4.6 Conclusion

L'interface développée avec Next.js constitue une **passerelle efficace entre l'utilisateur et l'intelligence artificielle embarquée**. Elle permet de vulgariser l'utilisation des LLMs et outils de vision artificielle via une interface intuitive. En renforçant ses fonctionnalités avancées, cette UI pourrait à terme s'adapter à une exploitation réelle à l'échelle métier (immobilier, finance, RH...).

5. Intégration continue et modularité

Chaque composant du projet a été conçu pour être autonome et testable séparément. Cela permet :

- D'adapter rapidement les modules (remplacer Gemini par LLaMA, par exemple)
- De tester un composant sans dépendre du front
- De maintenir un code propre, modulaire, et évolutif

Conclusion

Ce chapitre permet ainsi d'illustrer l'ensemble du cheminement technique mis en œuvre pour construire une solution complète de traitement de documents intelligents, alliant vision par ordinateur, extraction de texte, LLM et interface web ergonomique.

Conclusion Générale

Ce rapport retrace le travail effectué durant mon stage de fin d'études, réalisé au sein de l'entreprise **Yasmine Engineering Systems**, dans le cadre de l'obtention de mon **Master professionnel en Business Analytics** à l'ESPRIT School of Business. Cette expérience a représenté une opportunité précieuse pour mettre en pratique les compétences acquises au cours de ma formation, tout en découvrant les exigences du monde professionnel.

Le projet confié portait sur un enjeu concret et stratégique : **l'automatisation de la saisie des factures**. Dans un contexte où les volumes de documents à traiter sont croissants, la mise en place d'un système intelligent et automatisé permet non seulement de **réduire les délais de traitement**, mais aussi de **limiter les erreurs humaines** et d'**améliorer la fiabilité des processus comptables**.

Pour répondre à cet objectif, j'ai conçu une solution combinant plusieurs technologies de pointe : **la détection d'entités visuelles via des modèles de deep learning (YOLOv8)**, **l'OCR multilingue avec Tesseract**, ainsi que des modules d'**analyse sémantique** intégrant des modèles d'IA avancés. Cette approche a permis de construire une chaîne complète, capable d'extraire automatiquement les informations clés contenues dans les factures (dates, montants, émetteurs, etc.).

Les résultats obtenus sont prometteurs. Ils démontrent que l'automatisation peut significativement améliorer la productivité et la fiabilité du traitement documentaire, tout en ouvrant la voie à une exploitation intelligente des données extraites. Toutefois, des **perspectives d'amélioration** subsistent : l'extension de la solution à d'autres types de documents, l'intégration de modèles plus légers pour un déploiement en production, ou encore le renforcement des capacités de validation sémantique par IA.

En somme, ce projet m'a permis de développer mes compétences techniques en **data science**, **computer vision** et **déploiement de solutions intelligentes**, tout en m'initiant à la gestion de projet en entreprise. J'espère que le travail accompli contribuera à l'évolution des processus internes chez Délice Danone, et qu'il constituera une base solide pour de futures initiatives en automatisation documentaire.

Bibliographie

- [1] J. Han, M. Kamber et J. Pei, *Data Mining: Concepts and Techniques*, San Francisco: Morgan Kaufmann, 2012.
- [2] S. Institute, *SEMMA Methodology*, Cary: SAS Institute, 1998.
- [3] C.-D. Consortium, «Cross Industry Standard Process for Data Mining (CRISP-DM),» Bruxelles (ou « Brussels » si tu préfères anglais), NCR, DaimlerChrysler, SPSS, 1999.
- [4] K. Schwaber et J. Sutherland, «The Scrum Guide: The Definitive Guide to Scrum,» Scrum.org, 2020 (dernière version officielle).
- [5] C. M. Bishop, «Pattern Recognition and Machine Learning,» Springer, New York, 2006.
- [6] A. Krizhevsky, I. Sutskever et G. E. Hinton, « Imagenet classification with deep convolutional neural networks,» *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.
- [7] I. Goodfellow, Y. Bengio et A. Courville, *Deep Learning*, Cambridge: MIT Press, 2016.
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser et I. Polosukhin, «Attention is All You Need,» *Advances in Neural Information Processing Systems (NeurIPS)*, p. 5998–6008 , 2017.
- [9] J. Devlin, M.-W. Chang, K. Lee et K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» *Proceedings of NAACL-HLT (North American Chapter of the Association for Computational Linguistics – Human Language Technologies)*, 2019.
- [10] D. Jurafsky et J. H. Martin, *Speech and Language Processing* (3rd ed. Draft), Upper Saddle River: Prentice Hall / Pearson, 2013.
- [11] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora et e. al., «On the Opportunities and Risks of Foundation Models,» 2021.
- [12] R. Szeliski, *Computer Vision: Algorithms and Applications* (2nd ed.), Springer, 2022.

-
- [13] S. Ren, K. He, R. Girshick et J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,» *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.
- [14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu et A. C. Berg, «SSD: Single Shot MultiBox Detector,» *European Conference on Computer Vision (ECCV)*, pp. 21-37, 2016.
- [15] G. e. a. Jocher, «"YOLOv8: Ultralytics Official Implementation",» *arXiv preprint arXiv:2301.01234 (ou GitHub si vous citez le dépôt)*, 2023.
- [16] R. Smith, «"An Overview of the Tesseract OCR Engine",» 2007.
- [17] J. Al, «EasyOCR: Ready-to-use OCR with 80+ Supported Languages,» *GitHub repository*, 07 2020.
- [18] Y. C. C. L. W. e. a. Du, «PaddleOCR: An Open-Source Toolkit for Deep Learning Based Optical Character Recognition,» *arXiv preprint arXiv:2009.09941*, 09 2020.
- [19] FastAPI (Sebastián Ramírez), «FastAPI: Modern, Fast (High-performance), Web Framework for Building APIs with Python,» chez *GitHub repository*, 2019.
- [20] Streamlit Inc. (ou cocher Entreprise Auteur), « Streamlit: The fastest way to build data apps in Python,» *GitHub repository*, 2019.
- [21] Vercel Inc., «Next.js: The React Framework for Production,» *GitHub repository*, 2016.
- [22] Vercel Inc., « Vercel Official Website,» 2024.
- [23] D. Inc, « Docker: Empowering App Development with Containers,» *Official Documentation*, 2013.
- [24] Tzutalin, «Labellmg: Open-Source Graphical Image Annotation Tool,» *GitHub repository*, 2015.
- [25] Documentation officielle Ultralytics .