Projet — Application CV pour repérer et recommander des livres en bibliothèque

> Objectif : construire un prototype (desktop → mobile) qui, en pointant l'appareil photo vers des étagères, détecte des tranches de livres, redresse l'affichage (pour éviter de tourner la tête), extrait le titre/auteur/ISBN, récupère un résumé et propose des recommandations personnalisées qui apprennent de tes goûts.

---

1) Vue d'ensemble (architecture conceptuelle)

Flux principal (capture → résultat) :

1. **Capture image** (webcam / smartphone)
2. **Pré-traitement** (redressement, ajustement contraste, débrouillage bruit)
3. **Détection d'objets** (détecter les tranches / spines de livres)
4. **Extraction d'instances** (crop de chaque tranche détectée)
5. **Correction d'orientation** (détecter si le texte est vertical/horizontal, rotation)
6. **OCR** (extraire texte — titre, auteur, éventuellement code-barre / ISBN)
7. **Normalisation & matching** (nettoyage texte → rechercher métadonnées via API ou base locale)
8. **Récupération métadonnées** (résumé, catégories, couvertures depuis Open Library / Google Books / OPAC)
9. **Recommandation personnalisée** (score : correspondant au profil utilisateur)
10. **UI** (afficher résumé, indicateur « ça te plairait ? / pas intéressé »)
11. **Boucle d'apprentissage** (enregistrer feedback → mise à jour profil utilisateur)

Schéma simplifié :

```
Camera -> Preprocess -> Detector -> Crop & Rotate -> OCR -> Match -> Metadata ->
Recommender -> UI
```

---

2) Décisions importantes / contraintes à définir dès le départ

- **Prototype d'abord sur ordinateur** (facile à développer, debugger et entraîner les modèles). Mobile ensuite.
- **Traitement local vs serveur** : pour confidentialité et latence, tu peux opter pour traitement local (on-device) pour l'inférence; pour le training et les modèles lourds, serveur/GPU.
- **Données** : accès aux métadonnées via APIs publiques (Open Library, Google Books) → priorité pour l'MVP.
- **Budget et coûts** : LLMs ou API payantes (Google Vision, OpenAI) coûtent — prévoir si tu comptes générer des résumés via LLM.

- **Performances mobiles** : choisir des modèles légers (yolov8n, MobileNet, quantification) pour embarquer.

---

3) Phases (plan de développement pas à pas)

Phase 0 — Conception & recherche (préparatoire)

- Définis le MVP (features minimales) :
- détecter une tranche, extraire texte, retrouver résumé via API, afficher si tu aimes (feedback simple)
- Prépare l'environnement dev : Python, PyTorch, OpenCV, Tesseract, Streamlit (ou Flask) pour UI desktop.
- Récolte d'exemples photos dans des bibliothèques réelles (angles ±, éclairage variable).

Livrable : cahier des charges + dossier d'images brutes (~100–500 photos pour commencer).

Phase 1 — Proof-of-Concept (desktop) : OCR + matching

But minimal qui marche : 1. Prendre photo(s) d'étagères avec webcam ou smartphone. 2. Faire un pipeline simple : **prétraitement → OCR** (pytesseract / EasyOCR) sur l'image entière. 3. Tenter une **recherche textuelle** sur Open Library / Google Books par titre mot-clé. 4. Afficher les meilleurs résultats et le résumé. 5. Interface simple (Streamlit) qui montre les résultats en temps réel.

Ce que tu apprendras : OpenCV (prétraitement), OCR, appels API externes, UI simple.

Livrable : prototype desktop qui, pointé vers un livre unique ou une tranche claire, retrouve le livre et affiche résumé.

Phase 2 — Détection de tranches & orientation

- Annoter des images (bounding boxes pour chaque tranche de livre) — outils : LabelImg, CVAT, Roboflow.
- Convertir annotations en format COCO / YOLO.
- Entraîner un détecteur d'objets (YOLOv8 / Detectron2) pour trouver les spines.
- Pour chaque bbox : crop + estimer angle (orientation). Méthodes :
- heuristique via boîte englobante (ratio hauteur/largeur),
- ou détecter la direction du texte via CRAFT/EAST + orientation estimation,
- EasyOCR gère aussi les textes orientés.
- Appliquer rotation pour rendre le texte horizontal et plus lisible.

Livrable : pipeline qui détecte et redresse automatiquement chaque tranche.

Phase 3 — OCR robuste + matching amélioré

- Remplacer/combiner OCR : Tesseract (gratuit) ou EasyOCR (meilleur pour orienté), ou API Google Vision pour robustesse.
- Extraire entités : **titre**, **auteur**, **ISBN/barcode** (si visible) — utilises pyzbar pour lire les codes-barres.
- Nettoyage texte (noise, caractères parasites) → normalisation (lowercase, remove punctuation), tokenization.
- Matching :
- **Si ISBN trouvé** → match exact et récupération directe des métadonnées.
- **Sinon** → fuzzy matching (rapidfuzz) ou search via API (Open Library search) avec nettoyage.
- **Amélioration** : embeddings (sentence-transformers) pour similarité sémantique entre OCR text et titres/descriptions en base.

Livrable : matching fiable (>80% sur bons crops).

Phase 4 — Recommender & apprentissage des goûts

- **Approche initiale (simple & efficace)** : content-based
- Représenter chaque livre par un vecteur : embeddings du résumé + catégories/genres (concat).
- Profil utilisateur = moyenne pondérée des embeddings des livres "aimés" (update incrémental: moving average).
- Score = similarité cosinus(profile, livre).
- **Feedback** : bouton J'aime / Pas intéressé ; stocker ces interactions.
- **Améliorations** : LightFM (hybride contenu + facteurs latents), ou collaborative filtering (implicit) si tu as beaucoup d'utilisateurs.
- **Exploration vs exploitation** : proposer parfois des livres divers (ε-greedy) pour apprendre préférences nouvelles.

Livrable : système qui classe les livres selon l'affinité utilisateur et apprends en ligne.

Phase 5 — Résumés & assistants (optionnel)

- Priorité : afficher le résumé fourni par la base (OpenLibrary / Google Books).
- Option avancée : si pas de résumé, générer un résumé via un modèle de summarization (Hugging Face) ou un LLM (OpenAI). Attention aux coûts & latency.
- Présentation : afficher 1–3 phrases, tag genre, note moyenne si disponible.

Phase 6 — Portage mobile

- Choix techno : **React Native (Expo)** ou **Flutter** pour cross-platform.
- Intégration caméra + streaming image vers le back-end (ou inference on-device).
- Options :
- **On-device inference** : convertir modèle en ONNX → CoreML (iOS) / TFLite (Android) ; quantifier (int8) pour la vitesse.

- **Serveur** : envoyer image au serveur (FastAPI) qui renvoie résultats → plus flexible mais nécessite réseau.
- UI mobile : affichage AR-style léger : nom du livre flottant, résumé et boutons Like/Dislike.

Livrable : application mobile qui affiche en temps réel les informations pour une tranche scannée.

Phase 7 — Production & maintenance

- Containeriser (Docker), CI/CD (GitHub Actions), registry pour modèles (Weights & Biases / MLflow).
- Monitoring : logs d'ocr errors, taux de matching, latence, taux d'acceptation des recommandations.
- Collecte consentie de données pour ré-entrainement.

---

4) Choix techniques & bibliothèques recommandées

- Langage : **Python** pour la partie CV/ML, backend. Frontend : JS/TS (React / React Native) ou Flutter.
- CV & Détection : **OpenCV**, **PyTorch**, **Ultralytics YOLOv8**, **Detectron2**.
- OCR : **Tesseract (pytesseract)**, **EasyOCR**, **Google Vision API** (payant, mais performant).
- Text matching & NLP : **rapidfuzz** (fuzzy), **sentence-transformers** (embeddings), **faiss** (nearest neighbors), **transformers (Hugging Face)** pour summarization si besoin.
- Annotation : **LabelImg**, **CVAT**, **Roboflow** (gestion dataset + augmentation).
- Backend API : **FastAPI** (léger, moderne), **PostgreSQL** (metadata), **Redis** (cache, session), **MinIO / S3** (stockage images).
- UI desktop prototype : **Streamlit** (rapide), ou simple web React.
- Mobile : **React Native (Expo)** ou **Flutter**.

---

5) Données & annotation (conseils pratiques)

- **Collecte** : prendre photos variées : différents angles, éclairages, langues, étagères serrées/espacées.
- **Quantité minimale** pour un détecteur de spines utilisable : commencer avec ~1k–2k crops annotés (plus tu veux de robustesse plus il faut de données). Pour PoC, 200–500 images annotées peuvent suffire si tu utilises transfer learning.
- **Format** : annotations COCO ou YOLO. Conserver métadonnées (langue, éclairage, distance).
- **Augmentation** : rotation, flou, crop, changement de luminosité/contraste, bruits — Albumentations.

---

6) Évaluation — métriques à suivre

- Détection : **mAP@0.5** (mean Average Precision)
- OCR : taux de caractères corrects (CER) / taux de mots corrects (WER)
- Matching : précision@k, rappel@k
- Recommandation : precision@k, recall@k, NDCG, taux de clic (CTR) dans l'app
- Latence : temps moyen d'inférence (temps total de la capture à l'affichage)

---

7) Schéma de données (exemple simplifié)

table books - id - title - author - isbn - summary - embedding (vector) - cover_url - source

table users - id - profile_embedding - liked_books (history)

table interactions - user_id, book_id, action (like/dislike/view), timestamp

---

8) Exemple d'enchaînement (pseudocode très haut niveau)

```
# capture image
img = camera.capture()
# detect spines
bboxes = detector.predict(img)
for bbox in bboxes:
    crop = crop_box(img, bbox)
    crop = correct_orientation(crop)
    text = ocr(crop)
    isbn = read_barcode(crop)
    if isbn:
        meta = query_by_isbn(isbn)
    else:
        meta = search_by_text(text)
    embedding = embed(meta.summary)
    score = cosine_similarity(user.profile_embedding, embedding)
    show_result(meta, score)
```

---

9) Optimisation pour mobile (conseils)

- Utiliser des modèles compacts (yolov8n, mobilenetv3)
- Exporter vers ONNX → TFLite / CoreML
- Appliquer quantification (post-training quantization int8) et pruning
- Mesurer précision/latence sur device réel (emulateur ≠ réel)
- Gérer la mémoire et la caméra (batching frames, throttling inference pour 5–10 fps réalistes)

---

10) Problèmes fréquents & astuces

- **OCR bruyant** : prétraitement (CLAHE, binarization) et redressement améliorent énormément.
- **Spines collés** : détections confuses — augmenter dataset avec cas serrés.
- **Titres partiels** : utiliser embeddings de la phrase extraite pour meilleure correspondance.
- **Couleurs & reflets** : capter plusieurs images successives et fusionner résultats.

• **Langues** : Tesseract/EasyOCR gèrent plusieurs langues → charger lang appropriée.

---

11) Ressources & lectures recommandées (mises en pratique)

• Documentation Ultralytics (YOLOv8)
• Tutoriaux OpenCV (preprocessing & image ops)
• Tesseract / EasyOCR docs
• Hugging Face — sentence-transformers (embeddings)
• FastAPI tutorial
• Streamlit quick apps
• LabelImg / CVAT / Roboflow pour annotations

---

12) Checklist "Ce que tu fais dès maintenant" (MVP desktop)

1. Installer Python, créer un venv. Installer OpenCV, pytesseract, easyocr, requests, streamlit.
2. Prendre 100–300 photos d'étagères (varier angles/éclairages).
3. Faire un petit script : lire une image → OCR → recherche Open Library → afficher résumé.
4. Passer ensuite à l'annotation et entraînement du détecteur de spine.

---

13) Questions que ce plan résout / ce qu'on ne couvre pas (pour l'instant)

• *Fournit une route complète* depuis la capture jusqu'à la recommandation.
• *N'inclut pas* : code complet, scripts d'automatisation CI, UI mobile détaillée (mais donne les choix technologiques et étapes de portage).

---

14) Suivi possible (si tu veux aller plus loin)

• Atelier pratique (session pas-à-pas) : 1) PoC OCR + OpenLibrary, 2) annotation + entraînement YOLO, 3) pipeline complet et UI desktop, 4) portage mobile.
• Aide pour écrire les scripts d'entraînement, notebooks d'expérimentation, ou fichiers Docker.

Fin — Notes finales

J'ai cherché à rester pragmatique : commencer simple (OCR + matching via API), rendre cela robuste (détection + orientation), puis ajouter personnalisation et mobile. Le plan est modulaire — tu peux t'arrêter à n'importe quelle étape et avoir un produit utile.

Si tu veux, je peux maintenant : - te générer la checklist exacte pour la **Phase 1** (commandes d'installation, packages, et étapes concrètes) ; - ou bien préparer le template d'annotations (ex. structure COCO minimal) ; - ou encore, te proposer un prototype Streamlit de démonstration (sans mobile pour l'instant).

Dis ce que tu veux que je fasse ensuite et je te fournis tout en détail.