PROJET IAAV

Détection et interprétation en temps réel des gestes de la main par intelligence artificielle

1.Objectif

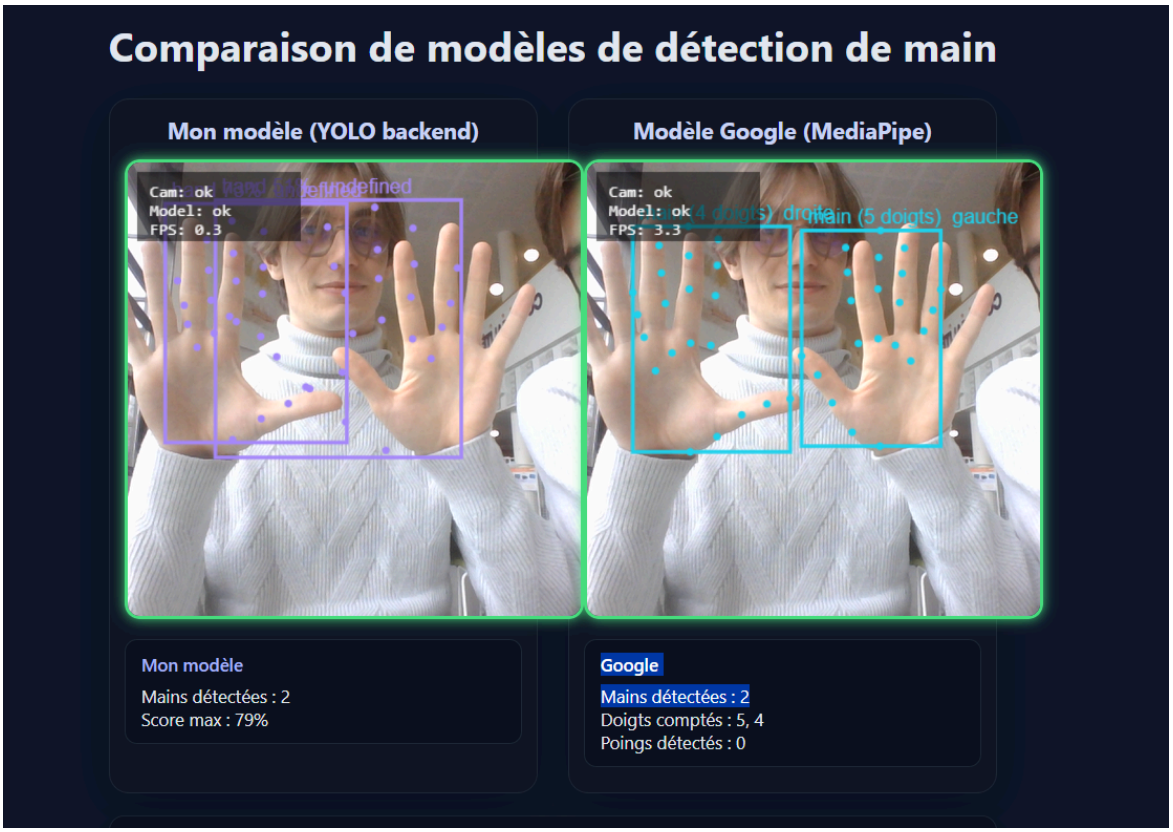
- Concevoir un système IA qui, en temps réel, détecte, reconnaît et interprète les gestes de la main à partir d'un flux vidéo.
- Sorties attendues : nombre de doigts (par main et total), posture (ouverte/poing/signes), direction du mouvement (gauche/droite/haut/bas), gestes dynamiques (fermeture progressive de doigts, prise d'objet).

2.Technologies utilisées

- Python + OpenCV : capture du flux caméra, prétraitement (redimensionnement, normalisation), orchestration du pipeline et du modèle IA.
- YOLO (ultralytics) , modèle avec des poids de base yolo11n permettant d'accélerer l'entrainement, permet de la détection rapide et est finetunable avec un dataset.
- MediaPipe Hands : extraction rapide et robuste des points clés de la main. MediaPipe est léger, temps réel et fournit 21 keypoints par main.
- React/JavaScript (front web) : interface temps réel pour afficher la vidéo, les overlays (zones de détection, trajectoires) et les infos détectées (doigts, geste, direction).
- fastapi (backend) : permet des calculs rapides un serveur en fond faisant tournée le modèle hand.pt et des requêtes API dédiés.

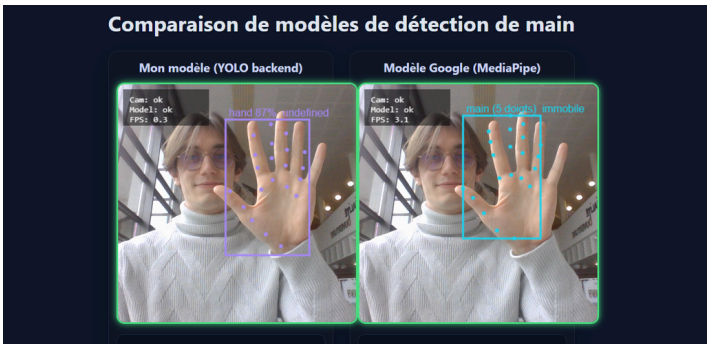
3.Création et fonctionnement de l'IA

- **Au départ:** Dataset maison : prises de vue des deux mains, variations de lumière/angles/distances, annotations précises (posture, direction, geste dynamique, nombre de doigts). Objectif : couvrir les cas réels de l'application.
- **Au final:** une portion du datasets freihand (https://lmb.informatik.uni-freiburg.de/resources/datasets/FreihandDataset.en.html) permettant d'avoir de très bonne annotations des conditions différentes et beaucoup plus d'images donc une meilleure IA, éviter le surraprentissage et la non diversité du dataset et ensuite la mauvaise annotations lorsque j'ai voulu créer un datasets a partir de plus d'images (via RobotFlow).
- Finetune de Yolo11n sur le datasets freihand avec 15 epochs avec un split du dataset 80/10/10
- Appel à MediaPipe (google) pour détecter les mains et les keypoints de la main.



4.Front : pourquoi deux écrans et comment ça fonctionne

- Écran 2 (tableau de bord texte/indicateurs) : affiche les infos structurées (compteur de doigts par main et total, posture reconnue, direction détectée, geste dynamique). Plus lisible pour valider la logique sans être pollué par la vidéo.
- Fonctionnement : le front reçoit en temps réel (WebSocket) les prédictions du backend et met à jour les overlays + les cartes d'état. React gère l'état global, un composant s'occupe de la vidéo/overlay, un autre des indicateurs.



5.Critique et performances

- Latence : il y a des gros problèmes de latence, on peut remarquer sur les images que mon modèle tourne à 0.3 fps alors que celui de google tourne à 3.6,
- Robustesse : tester sous différentes lumières, peaux, vitesses de geste, orientations de main. Les gestes dynamiques sont sensibles à la vitesse dû aux inférences. Le reste semble robuste. Le système mets du temps a se lancer.
- Objectif : Je n'ai pas mis en place de classes pour compter le nombre de doigt fautes de temps, pareil je n'ai pas réussi a détécter les gestes interactifs (fermeture de poing etc).

PARTIE TECHNIQUE

Architecture du projet:

```
hand-gesture-starter/
|
├─ package.json
├─ vite.config.js
├─ public/
├─ src/
|  ├─ main.jsx        ├─ App.jsx
|  ├─ index.css       ├─ styles.css
|  ├─ components/
|  |    ├─ CameraPanel.jsx
|  |    ├─ OverlayCanvas.jsx
|  |    └─ EventConsole.jsx
|  ├─ hooks/
|  |    ├─ useDetections.js
|  |    ├─ useBackendDetections.js
|  |    └─ useWebcam.js
|  ├─ detectors/
|  |    └─ FakeDetector.js
|  └─ assets/
|
├─ backend/
|  ├─ app.py          ├─ config.py
|  ├─ detector.py     ├─ schemas.py
|  ├─ requirements.txt
|  ├─ models/
|  |    ├─ hand.pt
|  |    └─ README.md
|  └─ README.md
|
└─ yolo_training/
   ├─ README.md
   ├─ prepare_freihand_yolo.py
   ├─ test_class.py
   ├─ FreiHAND_pub_v2.zip
   ├─ yolo11n.pt        ├─ yolo11n-pose.pt
   ├─ freihand_raw/
   └─ freihand_yolo/
      ├─ dataset.yaml
      ├─ images/ …   labels/ …
      └─ runs/
         └─ freihand/
            ├─ args.yaml
            ├─ results.csv
            └─ weights/
               ├─ best.pt
               └─ last.pt
```

FRONTEND

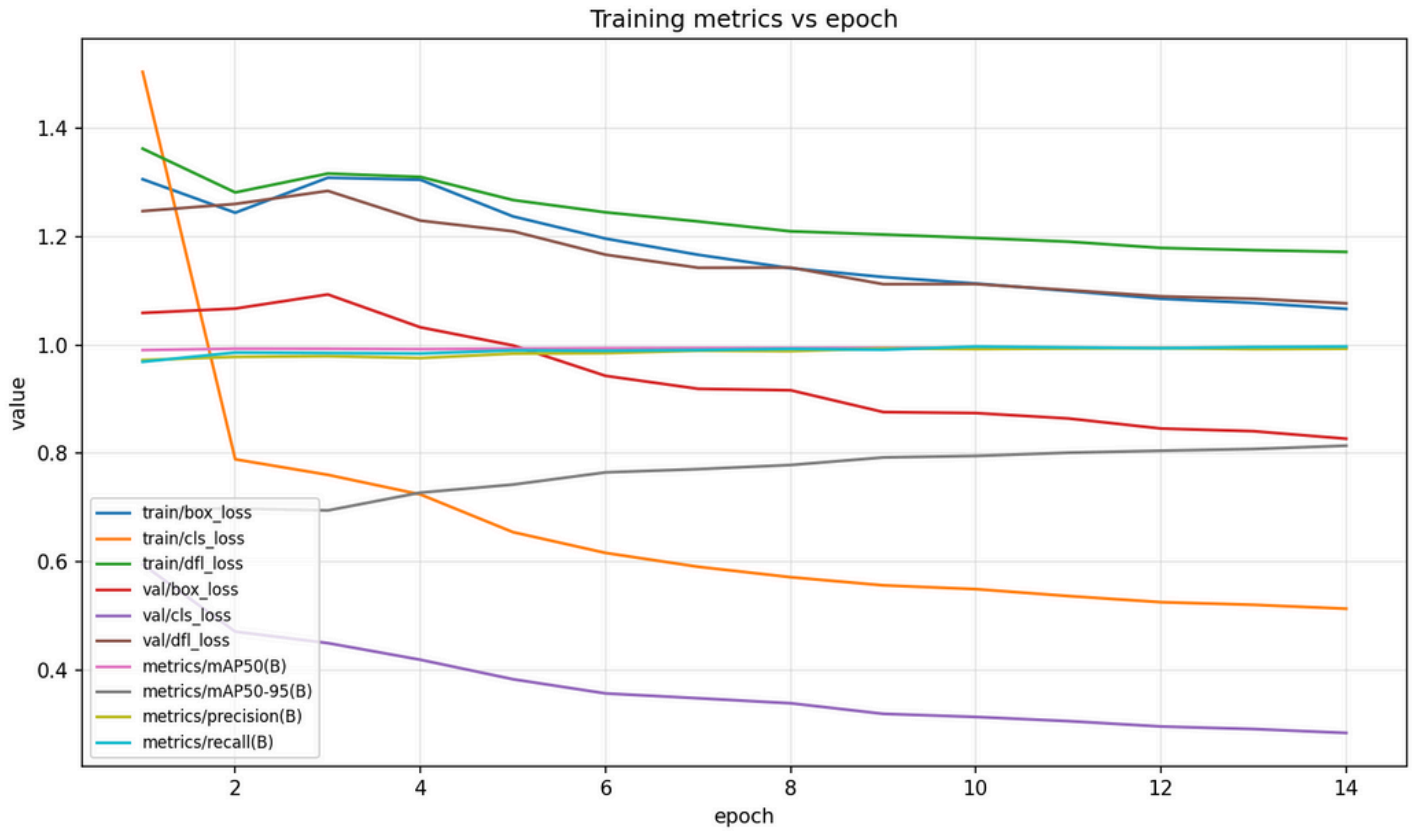
- Détection MediaPipe: src/hooks/useDetections.js + src/detectors/FakeDetector.js (MediaPipe HandLandmarker). Boucle rAF, mesure FPS, détecte doigts/poing/direction, génère des événements lisibles.
- Détection YOLO: src/hooks/useBackendDetections.js capture une frame vidéo, l'envoie en POST /detect (backend FastAPI, VITE_API_URL ou http://localhost:8000), mappe les résultats YOLO (score, label, couleur).

BACKEND

- Détection: backend/detector.py wrap Ultralytics YOLO (YOLOHandDetector.predict), filtrage optionnel des classes, normalisation des boxes, export des keypoints.
- Config: backend/config.py (env vars YOLO_MODEL_PATH, YOLO_DEVICE, YOLO_HAND_CLASSES), modèle par défaut backend/models/hand.pt. (permet de tester un autre modèle facilement il suffit de changer le chemin d'accès et qu'il soit en .pt)

ENTRAINEMENT.

- entrainement sur 15 epochs car déjà très long (4 heures d'entrainement) et donne de très bon résultat.
- on voit la courbe qui montre bien l'apprenstissage , la courbe loss qui descend bien.



SCRIPT DE CREATION HAND.PT

Extraction du dataset FreiHAND

1. crée le dossier freihand_raw/,
2. vérifie si les images sont déjà extraites,
3. extrait uniquement :
   - les images training/rgb/<id>.jpg,
   - les fichiers d'annotations training_xyz.json, training_K.json, training_scale.json.

Projection 3D → 2D pour créer les labels YOLO

1. FreiHAND fournit des joints 3D de la main.
2. Cette fonction utilise la projection matricielle pour convertir les points 3D en coordonnées 2D.
3. Elle calcule ensuite la bounding box normale YOLO en (cx, cy, w, h).

Keypoints (mode pose)

1. Convertit les **21 joints de la main** en points 2D,
2. normalise en [0,1],
3. renvoie une liste de keypoints compatibles avec YOLO.

Génération du dataset YOLO

copie l'image dans :

- images/train/,
- ou images/val/.

crée un fichier label .txt contenant :

- classe = 0 (main),
- bbox normalisée,

- keypoints si mode pose.

Création du fichier dataset.yaml

YOLO utilise dataset.yaml pour comprendre :

1. le chemin du dataset,
2. l'emplacement des images train/val,
3. le nombre de classes,
4. la forme des keypoints (actif uniquement en pose).

Entraînement YOLO

model = YOLO(args.model)

model.train(data=str(dataset_yaml), epochs=args.epochs, imgsz=args.img_size, batch=args.batch, device=args.device,

workers=args.workers, project=str(project), name=args.run_name, task=args.task,)

1. Charge un modèle YOLO pré-entraîné (yolo11n-pose.pt).
2. Lance un finetuning sur ton dataset FreiHAND transformé en YOLO.

Conclusion

Le projet aboutit à un pipeline complet et fonctionnel qui va de la capture vidéo à la détection en passant par l'interface web, reposant sur une intégration entre FastAPI, YOLO et une couche frontend en React/MediaPipe. Les acquis techniques sont : préparation du dataset FreiHAND au format YOLO, fine-tuning d'un modèle léger et performant, calcul d'événements gestuels côté client et visualisation en temps réel via overlays.

Les performances obtenues sont très encourageantes, avec une latence certes beaucoup moins performantes que le modèle de Google mais une excellente précision sur la validation (mAP50 ~0.995).De plus, l'architecture reste modulable, ce qui facilite l'évolution du projet.

Quelques axes d'amélioration émergent toutefois : poursuivre l'entraînement jusqu'au nombre d'époques prévu afin de stabiliser les métriques fines (mAP50-95), améliorer la pipeline de detection pour diminuer le temps entre l'envoie et la reception de l'image, structurer une première version de gestes dynamiques plus complexes, et renforcer la robustesse de l'expérience utilisateur via des indicateurs de confiance ou un léger lissage des prédictions, Améliorer le tracking pour qu'il soit plus persistent.

En résumé, le projet constitue une base solide pour un système de reconnaissance gestuelle en temps réel, performant et évolutif, prêt à être enrichi et évalué dans des conditions plus variées et pouvant être utilisé pour des utilisations basiques.