Université de **P**aris-**E**st **C**réteil (UPEC)
École **P**ublique d'**I**ngénieurs de la **S**anté **E**t du **N**umérique (EPISEN)
NeoXam

Analyse comparative des solutions de Computer Vision pour l'analyse de mouvements sportifs : Application au volley-ball

Présenté par Anais Messaoudene

anais.messaoudene@etu.u-pec.fr

Septembre 2025

Composition du jury :

Olivier Michel	Tuteur
Gwenhaël de La Provôté	Maître d'apprentissage

Remerciements

Avant tout, je souhaite remercier mes parents qui ont tout donné et sacrifié pour que je puisse réussir. Ils m'ont toujours encouragée à aller loin et à croire en mes capacités. Sans leur soutien, rien de tout cela n'aurait été possible.

Je pense aussi à mes cinq frères et sœurs, qui ont toujours été là pour moi, et qui représentent une source de force et de motivation au quotidien.

Je remercie également mes amis et mes camarades de promotion, avec qui nous avons toujours su nous entraider et garder une bonne ambiance, même dans les moments les plus intenses.

Je tiens à exprimer toute ma gratitude à mon maître d'apprentissage, qui m'a énormément bien accueillie dès le début, et qui s'est toujours rendu disponible pour m'accompagner et répondre à mes questions.

Enfin, un grand merci à mon tuteur, à l'ensemble des professeurs et responsables de l'école, pour la qualité de leur accompagnement, leur disponibilité et leurs précieux conseils qui m'ont permis de mener ce travail à bien.

À toutes et à tous, merci du fond du cœur.

Table des matières

Table des figures

Introduction

L'essor des technologies numériques a profondément transformé de nombreux secteurs, notamment celui du sport. Les outils de vision par ordinateur, en particulier, se révèlent être des solutions puissantes pour analyser les performances des athlètes, améliorer la préparation physique et optimiser les techniques. Ces technologies permettent de décortiquer des mouvements complexes, d'identifier des faiblesses techniques, et même de prévenir des blessures en détectant des postures incorrectes. Dans ce contexte, le volleyball, sport collectif qui nécéssite de la précision et une grande coordination, représente un domaine d'application pertinent pour ces outils.

La vision par ordinateur repose sur des algorithmes avancés capables d'identifier les points clés du corps humain, de mesurer les angles et de suivre les trajectoires en temps réel. Des outils comme OpenPose, BlazePose et MoveNet ont gagné en popularité en raison de leurs capacités à détecter et analyser les mouvements humains. Cependant, bien que ces outils aient été largement utilisés dans divers sports, leur efficacité dans un environnement spécifique comme le volleyball, où les mouvements sont rapides et complexes, reste à explorer en profondeur.

Cette étude se concentre sur une problématique pratique précise : quel outil d'estimation de pose humaine choisir pour analyser efficacement les mouvements de volleyball ? Contrairement aux travaux existants qui appliquent directement un outil spécifique à un sport donné – comme l'étude de Thibault NG (2023) utilisant OpenPose pour l'analyse d'escalade – notre approche vise à comparer systématiquement trois solutions majeures (OpenPose, BlazePose et MoveNet) sur des mouvements volleyball réels. L'objectif est d'identifier les forces et faiblesses de chaque outil selon la complexité des mouvements, en utilisant des vidéos capturées lors d'entraînements personnels et en mesurant les performances sur des critères objectifs tels que la précision de détection, le temps de traitement et la robustesse.

Ce mémoire est structuré en trois chapitres principaux. Le Chapitre 1 présente un état de l'art des technologies d'estimation de pose humaine, détaillant le fonctionnement commun de ces outils puis leurs différences spécifiques. Le Chapitre 2 expose en détail les architectures des trois systèmes étudiés (OpenPose, BlazePose et MoveNet), analysant leurs forces et limites structurelles pour l'analyse du volleyball. Le Chapitre 3 présente la méthodologie expérimentale utilisée, incluant la mise en œuvre des trois outils et le protocole de test sur les mouvements de volleyball, et enfin le Chapitre 4 analyse les résultats obtenus et compare les performances de chaque solution selon la complexité des gestes étudiés. Cette comparaison systématique permettra de fournir des recommandations pratiques pour le choix d'outil d'estimation de pose selon le contexte d'application sportive.

Chapitre 1

État de l'art

Objectifs :

- Définir les concepts fondamentaux de l'estimation de pose humaine
- Analyser les différentes familles d'architectures (Top-Down et Bottom-Up)
- Identifier les contraintes spécifiques au volleyball et leurs implications techniques

1.1 Introduction à l'estimation de pose humaine

L'estimation de pose humaine (*Human Pose Estimation*, HPE) consiste à localiser les articulations du corps humain dans des images ou des séquences vidéo, produisant un ensemble de *keypoints* représentant la posture à un instant donné. Ces informations permettent d'analyser les mouvements, la cinématique et les interactions avec l'environnement.

Depuis l'essor de l'apprentissage profond, les performances de la HPE ont considérablement progressé, rendant possible l'analyse de mouvements complexes, y compris dans des environnements dynamiques tels que les compétitions sportives.

Dans le cas du volleyball, discipline marquée par des actions rapides et explosives (smash, bloc, service), la HPE ouvre des perspectives intéressantes :

- Optimisation technique individuelle et collective
- Prévention des blessures par analyse biomécanique
- Assistance à l'arbitrage grâce à des mesures objectives
- Production de statistiques avancées pour le suivi des performances

Cependant, des limitations persistent face aux conditions extrêmes : occlusions fréquentes, changements rapides d'orientation, flou de mouvement, et grande amplitude articulaire.

1.2 Fondements théoriques

1.2.1 Systèmes de keypoints et annotations

Les systèmes d'annotation définissent un ensemble de points anatomiques représentatifs, tels que chevilles, genoux, hanches, épaules, coudes, poignets, yeux et oreilles.

Définition 1 — *Keypoints COCO* Le format COCO (*Common Objects in Context*) utilise 17 points de référence anatomiques standardisés pour représenter la pose humaine 2D.

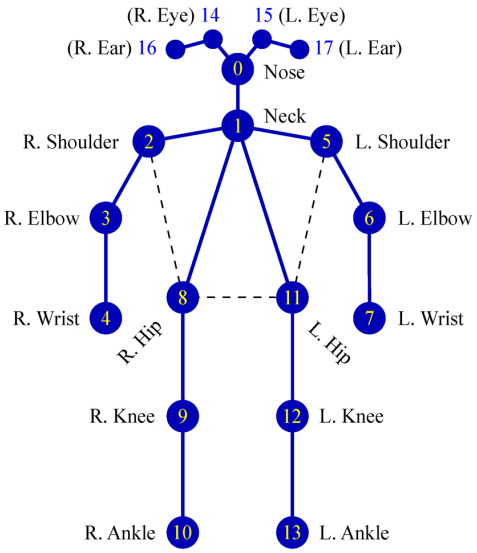

Figure 1 – Exemple d'annotation de 17 keypoints.

Définition 2 — *Keypoints BlazePose* BlazePose de Google propose 33 keypoints, incluant davantage de repères pour le tronc et les extrémités, permettant une analyse plus fine de la posture.

1.2.2 HPE 2D et HPE 3D

En *HPE 2D*, le réseau prédit la position des articulations dans le plan de l'image (u, v), avec un score de confiance par point. C'est léger et rapide, donc très adapté aux usages temps réel ou à l'analyse d'actions filmées par une caméra fixe. En revanche, tout ce qui relève de la profondeur reste ambigu : une même posture peut paraître différente selon l'angle de vue, et les auto-occlusions (bras devant le

torse, jambe masquée, etc.) dégradent la mesure. Les indicateurs biomécaniques (angles, amplitudes) héritent de cette dépendance au point de vue.

L'*HPE 3D* reconstruit les articulations dans l'espace (x, y, z)—via plusieurs caméras calibrées, des capteurs dédiés, ou un *lifting* 2D→3D monoculaire. On gagne en robustesse pour la cinématique (angles segmentaires, vitesses) et on s'affranchit du point de vue, mais au prix d'une mise en place plus lourde et d'un coût de calcul supérieur. En monoculaire, l'échelle absolue reste d'ailleurs incertaine.

En pratique, pour un suivi vidéo fluide et des métriques simples, la 2D suffit généralement. Dès que l'objectif est une analyse biomécanique fine et comparable entre séances ou sujets, la 3D devient préférable.

1.2.3 Architectures : Top-Down vs Bottom-Up

Les approches *Top-Down* procèdent en deux temps : d'abord la détection des personnes, puis, pour chaque région détectée, l'estimation de la pose mono-individu. Cette décomposition offre en général une très bonne précision par sujet—tant que les personnes sont de taille suffisante dans l'image—mais le coût croît avec le nombre d'individus et les petites silhouettes sont plus difficiles. C'est le parti pris de modèles comme MoveNet (mono-personne) ou BlazePose.

À l'inverse, les approches *Bottom-Up* prédisent tous les points clés de l'image d'un seul coup, puis les regroupent par individu à l'aide d'un mécanisme d'assemblage (par exemple les *Part Affinity Fields* d'OpenPose). Elles gèrent naturellement les scènes denses, car le coût dépend moins du nombre de personnes. En contrepartie, l'assemblage devient délicat lorsque les sujets se chevauchent fortement, et la précision peut être un peu moindre sur les articulations fines.

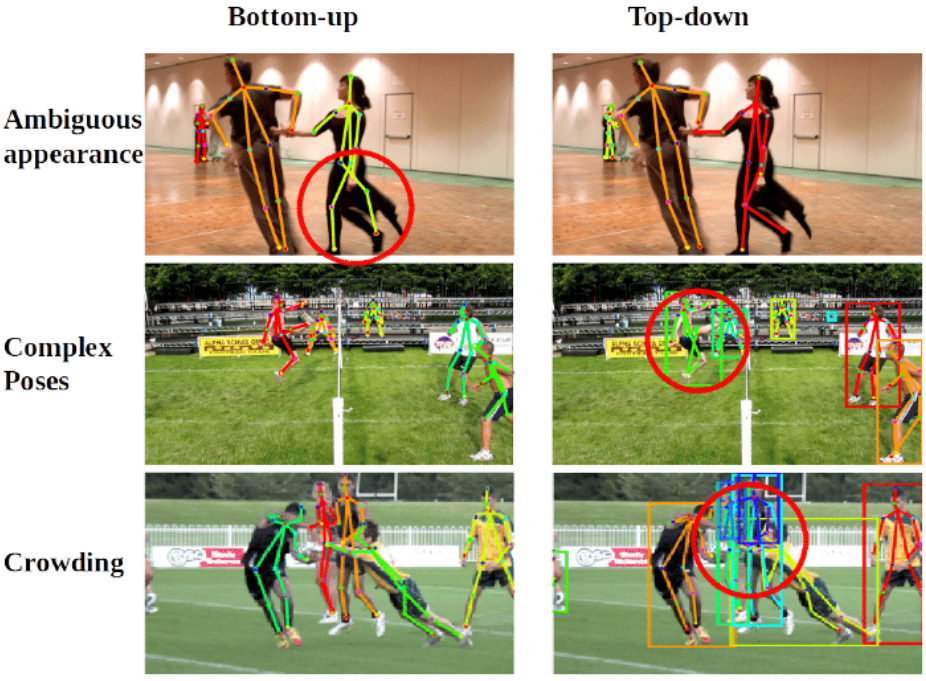

Figure 2 – Comparaison des paradigmes Top-Down et Bottom-Up.

1.2.4 Pipeline architectural standard

La plupart des architectures modernes partagent une structure commune.

Pipeline HPE standard : Les systèmes HPE modernes suivent généralement un pipeline en trois étapes :

1. Extraction de caractéristiques via un *backbone* (ResNet, MobileNet, VGG)
2. Tête de prédiction : production de heatmaps ou coordonnées (x, y, [z])
3. Post-traitement : reconstruction complète (ex. Part Affinity Fields dans OpenPose)

1.3 Évaluation et métriques

1.3.1 Datasets de référence

L'entraînement et l'évaluation des modèles reposent sur plusieurs jeux de données standards :

Dataset	Type	Nb. images	Keypoints
MPII Human Pose	2D	25k	16
COCO Keypoints	2D	200k	17
Human3.6M	3D	3.6M	32
Panoptic Studio	3D multi-view	65 caméras	15-20

Table 1.1 – Datasets de référence pour l'HPE.

1.3.2 Métriques de performance

L'évaluation des performances HPE repose sur plusieurs métriques standardisées :

Définition 3 — *PCK@t (Percentage of Correct Keypoints)* Pourcentage de keypoints correctement localisés dans un seuil de tolérance t :

$$PCK@t = \frac{\text{nb de keypoints corrects}}{\text{nb total de keypoints}} \times 100$$

Autres métriques importantes :

— **mAP** (mean Average Precision) utilisé dans COCO

— **OKS** (Object Keypoint Similarity) : mesure de la proximité pondérée entre prédiction et annotation

1.4 Défis spécifiques au volleyball

1.4.1 Contraintes techniques

Le volleyball pose des défis uniques pour l'estimation de pose :

- Mouvements explosifs et rapides (saut, smash, bloc)
- Occlusions fréquentes (filet, ballon, coéquipiers)
- Rotations rapides du tronc
- Amplitudes extrêmes (bras tendus, postures aériennes)

Figure 3 – Flou de mouvement et posture en extension lors d'une attaque au volleyball.

1.4.2 Limitations identifiées

Limites actuelles de la HPE en volleyball :

Les systèmes actuels d'estimation de pose présentent des limitations significatives face aux contraintes du volleyball, notamment :

- Sensibilité accrue aux occlusions en approche top-down
- Baisse de précision en bottom-up sur articulations distales
- Difficultés de traitement temps réel avec matériel limité
- Manque d'adaptations spécialisées pour les sports explosifs

Ces limitations résultent de la combinaison de facteurs techniques : la vitesse des mouvements génère du flou de mouvement dégradant la qualité des features extraites, les occlusions fréquentes perturbent les algorithmes de détection, et l'amplitude articulaire extrême sort du domaine d'entraînement des modèles standards.

1.5 Synthèse et problématique

L'état de l'art révèle que la HPE offre de réelles perspectives pour l'analyse du volleyball, mais les contraintes spécifiques de ce sport exposent des limites techniques importantes des architectures existantes.

Peu de comparaisons systématiques ont été menées sur les sports explosifs, et les adaptations sportives restent rares dans la recherche actuelle.

Problématique du mémoire :

Comment les différentes familles d'architectures de HPE réagissent-elles aux mouvements explosifs du volleyball, et quelles limitations techniques expliquent ces comportements ?

Cette question centrale guidera notre analyse comparative des approches Top-Down et Bottom-Up dans le contexte spécifique du volleyball.

Chapitre 2

Fondements des systèmes HPE pour l'analyse sportive

Ce chapitre analyse en détail les architectures techniques des trois systèmes d'estimation de pose humaine les plus répandus dans les applications sportives. L'objectif est de comprendre les mécanismes techniques sous-jacents qui déterminent leurs performances respectives en contexte volleyball, et d'identifier les sources potentielles de limitations pour l'analyse de mouvements explosifs.

Objectifs :

- Analyser les architectures OpenPose, BlazePose et MoveNet en détail

- Identifier les forces et limites structurelles de chaque approche
- Évaluer l'adéquation de ces outils pour l'analyse du volleyball

2.1 Introduction aux systèmes HPE modernes

L'analyse automatique du mouvement humain s'appuie aujourd'hui sur des modèles d'*Human Pose Estimation* (HPE) capables de détecter et de suivre des articulations clés dans une séquence vidéo. Dans le cadre de ce mémoire, notre objectif est de comparer plusieurs approches afin d'évaluer leur pertinence pour l'étude de mouvements sportifs, et en particulier ceux rencontrés dans la pratique du volley-ball.

La littérature regorge d'architectures et de variantes. Pour des raisons de représentativité et de faisabilité expérimentale, nous avons retenu trois solutions emblématiques :

1. **OpenPose**, pionnier des méthodes *bottom-up* introduisant les *Part Affinity Fields* (PAF) pour la reconstruction multi-personne

2. **BlazePose**, développé par Google, pensé pour les applications mobiles et le suivi précis du corps complet en temps réel

3. **MoveNet**, également proposé par Google, représentant une génération plus récente de modèles légers optimisés pour la rapidité et l'inférence sur des dispositifs contraints

Ces outils n'adoptent pas les mêmes choix architecturaux (approches *bottom-up* versus *top-down*, backbone lourds versus réseaux légers optimisés), ni les mêmes compromis entre précision et vitesse.

2.2 OpenPose : architecture bottom-up fondatrice

2.2.1 Contexte et innovation technique

OpenPose, développé au CMU Perceptual Computing Lab, a introduit en 2017 une approche *bottom-up* pour l'estimation de pose multi-personnes en temps (quasi) réel

Définition 4 — *Part Affinity Fields (PAF)* Les Part Affinity Fields sont des champs de vecteurs qui encodent, pour chaque type de membre (ex. épaule→coude), la direction et la cohérence de liaison entre deux articulations candidates.

Contrairement aux approches *top-down* (détecter les personnes puis estimer la pose par personne), OpenPose *détecte d'abord tous les points clés de la scène*, puis *résout un problème d'association* afin d'assembler des squelettes cohérents pour chaque individu.

Figure 4 – Exemple de détection multi-personnes par OpenPose.

2.2.2 Conventions d'annotation et sorties

OpenPose peut être entraîné/évalué sur différents schémas d'annotation :

- **COCO-17** : 17 points (standard détection 2D)
- **Body25** : 25 points (extension propriétaire OpenPose, utile pour tronc/bassin)
- Extensions visage/mains/pieds disponibles dans la distribution OpenPose

Sorties OpenPose : Chaque image RGB $I \in \mathbb{R}^{H \times W \times 3}$ produit :

1. un jeu de **cartes de chaleur** $\{\mathbf{S}_k\}_{k=1..K}$, K étant le nb de keypoints
2. un jeu de **PAF** $\{\mathbf{L}_c\}_{c=1..C}$, C étant le nb de types de membres

2.2.3 Architecture multi-étapes

Pipeline OpenPose multi-stage : Le réseau suit un schéma *multi-stage* avec supervision intermédiaire :

1. **Backbone** Φ (VGG-19 à l'origine) extrait des *feature maps* $\mathbf{F} = \Phi(I)$
2. **Étape 1** : deux têtes convolutionnelles produisent $\mathbf{S}^{(1)}$ (heatmaps) et $\mathbf{L}^{(1)}$ (PAF)
3. **Étapes** $t>1$: chaque étape reçoit la concaténation $[\mathbf{F}, \mathbf{S}^{(t-1)}, \mathbf{L}^{(t-1)}]$ et sort $\mathbf{S}^{(t)}, \mathbf{L}^{(t)}$

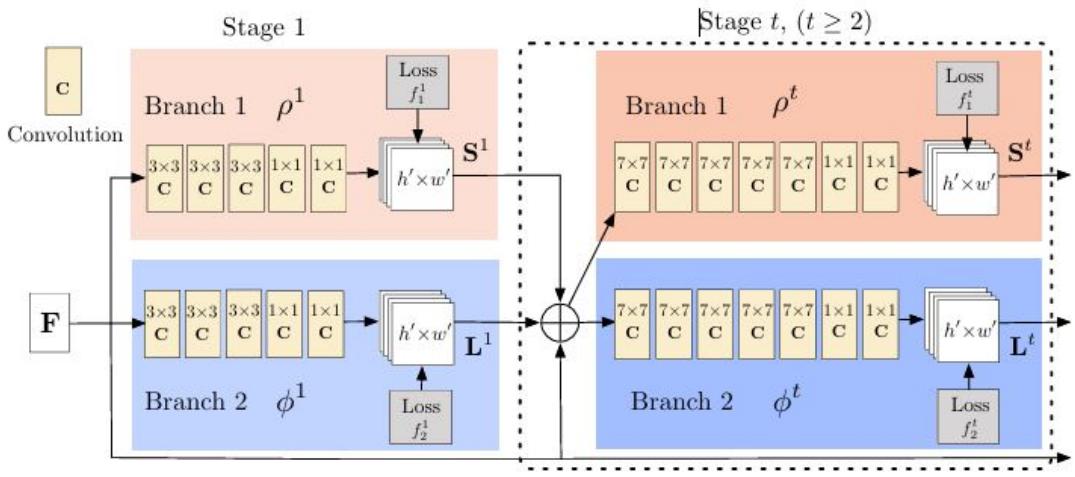

Figure 5 – Architecture multi-stage d'OpenPose avec supervision intermédiaire.

Cette itération raffine progressivement la localisation (heatmaps) et la cohérence de liaison (PAF). À l'entraînement, *chaque* étape est supervisée :

$$\mathcal{L} \;=\; \sum_{t=1}^{T} \Big(\lambda_S \sum_{k=1}^{K} \|\mathbf{S}_k^{(t)} - \mathbf{S}_k^{\star}\|_2^2 \;+\; \lambda_L \sum_{c=1}^{C} \|\mathbf{L}_c^{(t)} - \mathbf{L}_c^{\star}\|_2^2 \Big)$$

où $(\mathbf{S}^{\star}, \mathbf{L}^{\star})$ sont les cibles, et T est le nombre d'étapes (typ. $T{=}6$).

2.2.4 Génération des labels et décodage

Heatmaps. Pour chaque articulation annotée de position $\mathbf{p}_k$, on génère une carte cible gaussienne :

$$\mathbf{S}_k^{\star}(\mathbf{x}) \;=\; \exp\Big(- \frac{\|\mathbf{x} - \mathbf{p}_k\|_2^2}{2\sigma^2} \Big)$$

PAF. Pour un membre c reliant $(\mathbf{p}_a, \mathbf{p}_b)$, on définit le vecteur unitaire :

$$\mathbf{v}_c \;=\; \frac{\mathbf{p}_b - \mathbf{p}_a}{\|\mathbf{p}_b - \mathbf{p}_a\|_2}$$

Algorithm 1: Décodage PAF (simplifié)

[1] $\{\hat{\mathbf{p}}_{k,i}\} \leftarrow$ NMS sur $\{\mathbf{S}_k\}$ candidats par articulation chaque membre $c = (a, b)$ Calculer $s_{ij}^{(c)}$ pour toutes paires $(\hat{\mathbf{p}}_{a,i}, \hat{\mathbf{p}}_{b,j})$ via intégrale PAF $M_c \leftarrow$ matching biparti max sur $s_{ij}^{(c)}$ (seuil de validité) Fusionner $\{M_c\}_c$ pour construire des graphes (squelettes par individu) **return** ensembles de poses $\{\Pi_m\}$

2.2.5 Analyse critique pour le volleyball

Limitations OpenPose en volleyball :

OpenPose présente des limitations structurelles face aux contraintes du volleyball :

- Coût computationnel élevé (stages multiples, PAF + heatmaps)
- Assemblage sensible aux occlusions sévères (filet, bras croisés)
- Scaling défavorable quand beaucoup d'individus sont proches

Le volleyball combine *vitesses élevées*, *occlusions fréquentes* et *changements d'orientation* brusques. En situation de rallye, la présence simultanée de plusieurs joueurs et d'occlusions croisées dégrade l'assemblage PAF. Lors des mouvements explosifs, la latence élevée sur CPU empêche une analyse en temps réel.

2.3 BlazePose : architecture optimisée mobile

2.3.1 Approche à deux étages

BlazePose est une architecture développée par Google Research, conçue spécifiquement pour le suivi en temps réel de la posture humaine sur des dispositifs mobiles

Définition 5 — *Pipeline BlazePose* BlazePose adopte une stratégie à **deux étages** combinant :

1. Un détecteur rapide basé sur une variante optimisée de BlazeFace
2. Un réseau de régression de landmarks prédisant directement les coordonnées

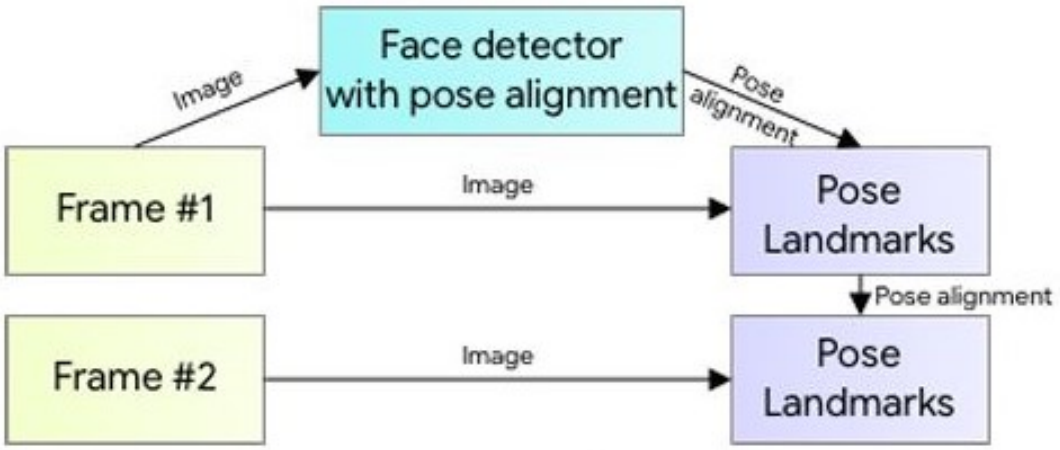

Figure 6 – Pipeline BlazePose : détection rapide suivie d'un réseau de régression de landmarks.

2.3.2 Topologie étendue de 33 points

Avantage topologique BlazePose : L'extension de la topologie à **33 points clés** couvre non seulement les articulations principales (comme en COCO), mais également le visage, les mains et les pieds, permettant une analyse biomécanique plus fine.

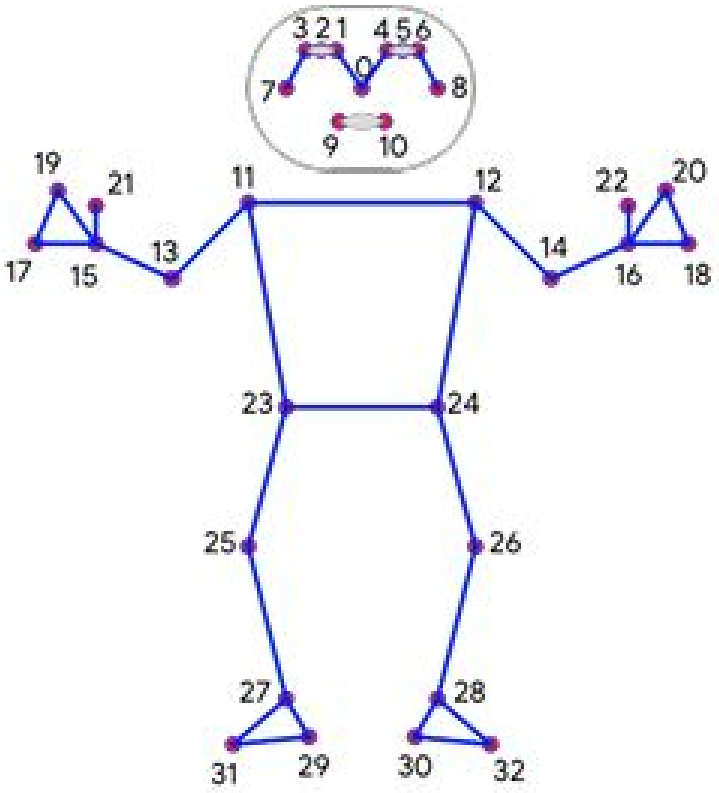

Figure 7 – Topologie BlazePose (33 landmarks) incluant visage, tronc et extrémités.

Dans le cas du volley-ball, cela ouvre la possibilité d'étudier la coordination entre les segments

distaux (poignets, chevilles) et proximaux (épaules, hanches), notamment lors du service et du smash.

2.3.3 Performances et contraintes

Performances BlazePose : Les auteurs rapportent que BlazePose atteint une vitesse **>30 FPS sur smartphone**, contre quelques FPS pour OpenPose dans les mêmes conditions, mais avec des limitations :

- Robustesse multi-personnes faible
- Précision réduite en conditions d'occlusion
- Dépendance à la qualité initiale de détection

2.4 MoveNet : architecture single-shot optimisée

2.4.1 Approche single-shot

MoveNet est une architecture plus récente développée par Google TensorFlow (2021), conçue pour offrir une estimation de pose en **temps réel** sur une large gamme de périphériques.

Définition 6 — *Approche MoveNet* Contrairement à OpenPose (Bottom-Up) et BlazePose (Top-Down en deux étapes), MoveNet adopte une approche **single-shot Top-Down**, où la détection et la régression des keypoints sont réalisées en une seule passe réseau.

2.4.2 Variantes et architecture

Variantes MoveNet : MoveNet existe en deux variantes principales :

- **Lightning**, optimisée pour la vitesse, capable de tourner à plus de 50 FPS sur CPU
- **Thunder**, plus précise mais légèrement plus lente, orientée vers des tâches nécessitant une meilleure robustesse

Caractéristique	MoveNet Lightning	MoveNet Thunder
Résolution d'entrée	192 × 192 pixels	256 × 256 pixels
Backbone	MobileNetV2 (50% canaux)	MobileNetV2 complet
Résolution heatmaps	48 × 48	64 × 64
Temps d'inférence	< 20ms (ARM Cortex-A78)	40-80ms (selon plateforme)
Précision PDJ@0.5	~75% (COCO)	~81% (COCO)
Cas d'usage	Temps réel strict	Compromis précision/vitesse

Figure 8 – Les deux variantes de MoveNet : Lightning (rapide) et Thunder (précise).

Le cœur de MoveNet repose sur un **backbone MobileNetV3 optimisé**, qui extrait des descripteurs visuels légers mais efficaces. Ces descripteurs alimentent un *single-shot regression head*, capable de prédire directement les coordonnées des 17 keypoints.

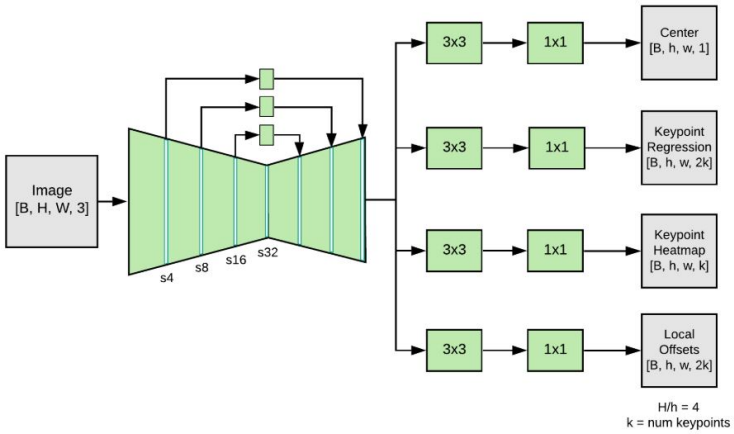

Figure 9 – Architecture MoveNet.

2.4.3 Évaluation comparative pour le volleyball

Adéquation MoveNet au volleyball :

Dans le contexte du volley-ball, MoveNet répond particulièrement à la contrainte du **temps réel**, mais présente des limitations techniques :

- Analyse possible des séquences rapides (service sauté, attaque)
- Latence très faible permettant l'intégration temps réel
- Nombre limité de keypoints (17) réduisant la finesse biomécanique
- Absence de gestion multi-personnes native

2.5 Synthèse comparative

2.5.1 Tableau comparatif des architectures

La présentation détaillée des trois modèles étudiés met en évidence des philosophies de conception distinctes et des compromis techniques différenciés.

Table 2.1 – Résumé comparatif des trois outils étudiés

Critère	OpenPose	BlazePose	MoveNet
Approche	Bottom-up, multi-personnes	Top-down, orienté individuel	Top-down, optimisé latence
Architecture	CNN multi-étapes + PAF	CNN léger + 33 key-points	Réseau léger optimisé TensorFlow Lite
Nombre de points	17 (COCO), 25 (Body25)	33 (corps complet)	17 (COCO-like)
Précision (AP/PCK)	Élevée, mais variable selon occlusions	Bonne précision articulations fines	Bonne sur postures courantes, sensible aux cas extrêmes
Vitesse	Lente sans GPU	Temps réel mobile/-desktop	Temps réel sur CPU/GPU et mobile
Robustesse aux occlusions	Moyenne (assemblage sensible)	Faible (fortement dépendant détection initiale)	Moyenne, sensible aux chevauchements
Adéquation volley-ball	Bonne pour scènes multi-joueurs	Pertinente pour analyse individuelle	Adaptée aux analyses rapides

2.5.2 Tendances architecturales

Compromis précision-vitesse : L'analyse révèle une tendance générale : **plus un modèle est léger et rapide, plus il sacrifie une partie de la précision ou de la robustesse dans des conditions complexes comme celles rencontrées en volley-ball**. À l'inverse, des architectures plus lourdes comme OpenPose offrent une meilleure cohérence anatomique, mais au prix d'une latence importante.

Ces constats orientent directement la suite de notre étude : le Chapitre 3 présentera l'évaluation empirique de ces modèles sur des séquences de volley-ball réelles, afin de vérifier dans quelle mesure les limites théoriques identifiées ici se traduisent en résultats concrets sur le terrain.

Conclusion du Chapitre

Cette analyse détaillée des trois architectures HPE révèle des approches fondamentalement différentes pour répondre aux défis de l'estimation de pose en temps réel. Chaque système présente des avantages spécifiques mais aussi des limitations qui anticipent les difficultés rencontrées dans l'analyse du volleyball.

Les constats théoriques établis dans ce chapitre guideront l'évaluation expérimentale à venir, où nous vérifierons empiriquement ces prédictions sur des séquences réelles de volleyball.

Chapitre 3

Méthodologie expérimentale

Ce chapitre présente la méthodologie adoptée pour l'expérimentation et l'évaluation des outils d'estimation de pose humaine retenus dans le cadre de ce mémoire, à savoir OpenPose, BlazePose et MoveNet. Après avoir décrit les gestes sportifs étudiés, nous détaillons le protocole de collecte des données vidéo, l'environnement matériel et logiciel mis en place, ainsi que la procédure suivie pour l'implémentation de chaque outil. Enfin, les critères d'évaluation retenus sont explicités, afin de préparer l'analyse comparative qui sera présentée dans le chapitre suivant.

Objectifs :

- Décrire les mouvements sportifs analysés et leur pertinence pour l'étude.
- Présenter le protocole de collecte et d'organisation des données vidéo.
- Documenter l'installation et la configuration des outils HPE.
- Définir les critères d'évaluation en lien avec la problématique.

3.1 Description des mouvements étudiés

L'étude expérimentale porte sur trois gestes fondamentaux du volley-ball : le service, la réception et l'attaque. Ces mouvements ont été choisis car ils présentent des caractéristiques cinématiques contrastées, permettant ainsi d'évaluer les performances des systèmes d'estimation de pose dans des contextes variés.

Le **service** constitue l'un des gestes de base du volley-ball. Il consiste à lancer le ballon puis à le frapper afin de le faire franchir le filet vers le camp adverse. Ce geste implique une coordination des membres supérieurs et inférieurs, et représente un mouvement relativement fluide mais rapide.

La **réception**, en revanche, est un geste défensif essentiel. Elle consiste à contrôler et rediriger le ballon après une attaque adverse, en adoptant une posture stable, jambes légèrement fléchies et bras tendus. La qualité de la réception conditionne directement la poursuite du jeu et nécessite une précision posturale que les systèmes HPE doivent pouvoir détecter.

Enfin, l'**attaque** (ou smash) est un geste explosif visant à frapper le ballon en suspension pour tenter de marquer un point. Ce mouvement combine un saut vertical, une extension rapide du bras dominant et une rotation du tronc. Sa dynamique en fait un test exigeant pour évaluer la robustesse des outils face à des mouvements rapides.

Afin de disposer d'un corpus représentatif, plusieurs vidéos longues comprenant une vingtaine de répétitions par geste ont été enregistrées. Pour l'analyse, un segment représentatif de chaque geste a été sélectionné, constituant un cycle complet utilisable pour les trois outils.

3.2 Collecte et préparation des données vidéo

Les vidéos ont été enregistrées dans le gymnase Alice Milliat à Choisy-le-Roi, dans des conditions lumineuses stables. Le dispositif de captation reposait sur un iPhone 16 Pro Max fixé sur trépied, permettant d'assurer une bonne stabilité ainsi qu'une résolution adaptée à l'analyse.

Chaque séquence a été filmée en Full HD (1920×1080) à 30 images par seconde, encodée en H.264. Pour chaque type de geste, une vidéo distincte a été isolée et placée dans un répertoire spécifique, garantissant une organisation claire des données. Aucun marqueur visuel ni matériel complémentaire n'a été utilisé, afin de conserver des conditions réalistes d'utilisation.

Les vidéos obtenues constituent ainsi un jeu de test homogène, suffisamment exigeant pour évaluer la capacité des modèles à détecter des articulations dans des gestes rapides, tout en restant proches des conditions réelles d'analyse sportive.

3.3 Environnement expérimental

Les expérimentations ont été réalisées sur une machine virtuelle Ubuntu 22.04 hébergée sur l'infrastructure VMware vSphere de l'école. Cette VM disposait de quatre vCPU, de 16 Go de mémoire vive et de 80 Go de disque. Aucun GPU n'étant disponible, l'ensemble des tests a été effectué en mode CPU, ce qui constitue une contrainte supplémentaire mais également une situation réaliste pour certains cas d'usage.

L'environnement logiciel reposait sur Python 3.10.12 et incluait notamment les bibliothèques *OpenCV* (v4.11.0.86), *Numpy* (v1.26.4), *Mediapipe* (v0.10.21 pour BlazePose) et *TensorFlow Hub* pour l'implémentation de MoveNet. OpenPose a été compilé à partir du dépôt officiel GitHub (branche master, 2024).

Cette configuration a été choisie de manière à garantir la reproductibilité des résultats, tout en permettant de mettre en évidence l'impact des ressources limitées sur la performance des différents

outils.

3.4 Mise en œuvre des outils

Pour chacun des trois outils étudiés, un script `Python` a été développé afin d'automatiser le traitement des vidéos, la sauvegarde des sorties (vidéos annotées et/ou JSON) et la collecte de métriques (temps moyen par frame, écart-type du temps, taux de détection). Les **scripts complets** sont fournis en **Annexe A** : A.1 OpenPose, A.2 BlazePose, A.3 MoveNet.

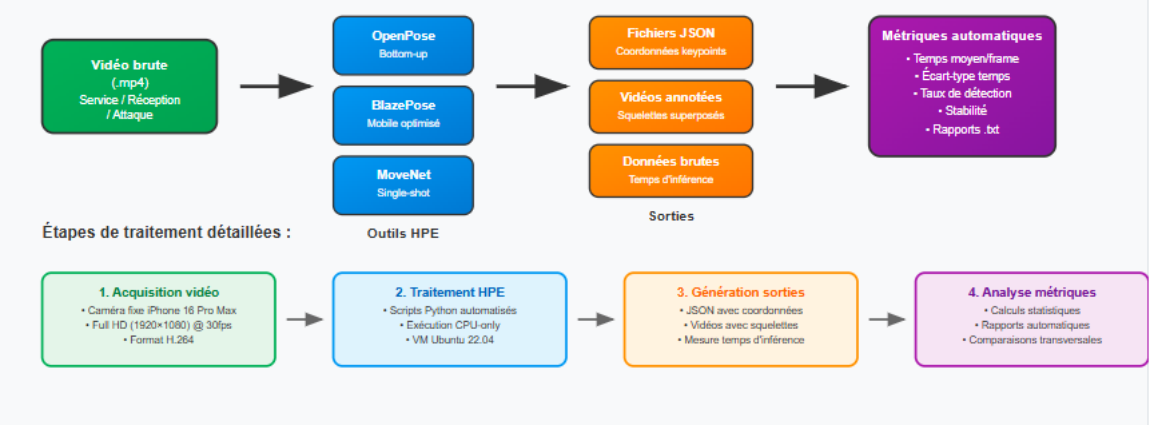

Figure 10 – Pipeline expérimental : de la vidéo brute (`.mp4`) aux outils HPE, puis au calcul automatique des métriques.

3.4.1 OpenPose

Le processus suivi avec OpenPose se décompose en quatre étapes : *(i)* lecture de la vidéo, *(ii)* génération de JSON par frame, *(iii)* création d'une vidéo annotée, *(iv)* analyse des JSON pour dériver les métriques. Ci-dessous, l'appel au binaire depuis Python (extrait). *Script complet :* Annexe A.1.

```
subprocess.run([
  OPENPOSE_BIN,
  "--video", VIDEO_PATH,
  "--write_json", OUTPUT_JSON_DIR,
  "--display", "0", "--render_pose", "1",
  "--write_video", OUTPUT_VIDEO, "--model_folder", MODEL_FOLDER
])
```

Après l'exécution, les JSON sont parsés pour compter la détection par frame (extrait) :

```
json_files = sorted(f for f in os.listdir(OUTPUT_JSON_DIR) if f.endswith(".json"))
detection_flags = []
for fn in json_files:
    with open(os.path.join(OUTPUT_JSON_DIR, fn), "r") as f:
        data = json.load(f)
    detection_flags.append(1 if data.get("people") else 0)
# detection_rate = 100 * np.mean(detection_flags)
```

3.4.2 BlazePose

BlazePose (MediaPipe) traite la vidéo image par image et produit directement l'annotation du squelette. Un **seuil de visibilité** $\theta = 0{,}2$ est appliqué pour considérer un keypoint comme valide ; une frame est dite « détectée » si au moins 5 keypoints dépassent ce seuil. *Script complet :* Annexe A.2.

```
mp_pose = mp.solutions.pose
pose = mp_pose.Pose(
  static_image_mode=False,
   min_detection_confidence=0.5,
  model_complexity=2
)
THRESHOLD = 0.2
```

```
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
start = time.time(); results = pose.process(frame_rgb); end = time.time()
inference_times.append((end - start) * 1000)

if results.pose_landmarks:
    detected = sum(1 for lm in results.pose_landmarks.landmark
                   if lm.visibility > THRESHOLD)
    if detected >= 5: detected_frames += 1
# detection_rate = 100 * detected_frames / frame_count
```

3.4.3 MoveNet

MoveNet *SinglePose/Thunder* est chargé depuis TensorFlow Hub ; chaque frame est redimensionnée (`resize_with_pad`) en 256×256 avant l'inférence. Le comptage « frame détectée » suit le même principe (≥ 5 keypoints avec confiance $> \theta$). *Script complet :* Annexe A.3 .

```
MODEL_URL = "https://tfhub.dev/google/movenet/singlepose/thunder/4"
model = hub.load(MODEL_URL)
movenet = model.signatures['serving_default']
THRESHOLD = 0.2
```

```
img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
img_resized = tf.image.resize_with_pad(tf.convert_to_tensor(img_rgb), 256, 256)
input_tensor = tf.expand_dims(tf.cast(img_resized, dtype=tf.int32), axis=0)
outputs = movenet(input_tensor)  # keypoints + confiances
```

3.5 Critères d'évaluation

Afin de comparer les trois outils dans des conditions homogènes, plusieurs critères ont été retenus. Le premier est la **performance computationnelle**, mesurée par le temps moyen de traitement par frame. La **fiabilité de la détection** est ensuite évaluée à travers le taux de frames correctement annotées et la présence des keypoints attendus. La **stabilité** est estimée par l'écart-type des temps d'exécution, traduisant la régularité des performances.

Un critère complémentaire concerne la **facilité de mise en œuvre**, appréciée qualitativement à travers la simplicité d'installation et la compatibilité avec des environnements de recherche. Enfin, une évaluation qualitative de la **robustesse face aux mouvements rapides** complète l'analyse, en lien direct avec la problématique de ce mémoire.

3.6 Conclusion intermédiaire

Ce chapitre a présenté la méthodologie expérimentale mise en place pour comparer OpenPose, BlazePose et MoveNet dans le cadre de l'analyse de mouvements sportifs en volley-ball. Les choix opérés concernant les gestes étudiés, la collecte vidéo, l'environnement logiciel et les critères d'évaluation visent à établir un protocole rigoureux et reproductible.

Le chapitre suivant sera consacré à la présentation des résultats obtenus et à leur analyse comparative, afin de mettre en évidence les atouts et les limites de chaque solution.

Chapitre 4

Analyse des résultats et discussion

Ce chapitre répond directement à la problématique posée en introduction : *« Comment les différentes familles d'architectures d'estimation de pose humaine (HPE) réagissent-elles aux mouvements explosifs du volleyball, et quelles limitations techniques expliquent ces comportements ? »* Nous présentons d'abord les résultats expérimentaux (temps de traitement, stabilité, taux de détection), puis nous discutons les tendances observées à la lumière des caractéristiques architecturales exposées au Chapitre 2. Les scripts complets ayant permis de produire ces mesures sont fournis en **Annexe A** — A.1 OpenPose, A.2 BlazePose, A.3 MoveNet.

Objectifs :

- Présenter et structurer les résultats bruts obtenus sur *service*, *réception* et *attaque*.
- Analyser comparativement OpenPose, BlazePose et MoveNet au regard de leurs architectures (Chap. 2).
- Discuter les limites de l'étude et proposer des perspectives concrètes.

4.1 Résultats expérimentaux

Les mesures suivantes ont été obtenues **sur CPU uniquement** (VM Ubuntu 22.04, 4 vCPU, 16 Go RAM), conformément à la méthodologie décrite au Chapitre 3. Le **taux de détection** pour BlazePose et MoveNet correspond à la proportion d'images où *au moins cinq keypoints* dépassent un **seuil de confiance** $\theta = 0{,}2$ (voir scripts A.2 et A.3). Pour OpenPose, la détection par image est dérivée de la présence d'au moins une entrée dans `people[]` des JSON (voir A.1).

Table 4.1 — Synthèse globale (moyenne sur les trois gestes)

Outil	Temps moy. (ms/frame)	Écart-type (ms)	Détection (%)	Précision qualitative
OpenPose (CPU)	**82 819,09**	*2 485*	**100,0**	Fine sur membres supérieurs ; très lent
BlazePose	**134,18**	51,15	70,27	Variable ; pertes en attaque (bras/mains)
MoveNet (Thunder)	**64,79**	52,13	**100,0**	Bon compromis ; extrémités parfois manquantes

Figure 11 – Synthèse globale (moyenne sur les trois gestes).

Notes : (i) moyennes arithmétiques sur Service/Réception/Attaque ; (ii) le seuil $\theta = 0{,}2$ est celui utilisé dans les scripts A.2 et A.3.

4.1.1 Service

Outil	Frames	Temps moyen (ms)	Écart-type (ms)	Taux détection (%)
OpenPose (CPU)	123	83 287,76	*2 499*	100,0
BlazePose	123	131,44	58,69	69,9
MoveNet (Thunder)	123	62,69	48,05	100,0

Table 4.1 – Service : métriques par outil (rapports de scripts).

Figure 12 – Service — comparaison visuelle sur la même frame annotée par OpenPose (a), BlazePose (b) et MoveNet (c). Contexte CPU ; seuil $\theta = 0{,}2$ pour BlazePose/MoveNet.

4.1.2 Réception

Outil	Frames	Temps moyen (ms)	Écart-type (ms)	Taux détection (%)
OpenPose (CPU)	98	81 855,19	*2 456*	100,0
BlazePose	98	173,74	28,36	99,0
MoveNet (Thunder)	98	61,83	52,26	100,0

Table 4.2 – Réception : métriques par outil (rapports de scripts).

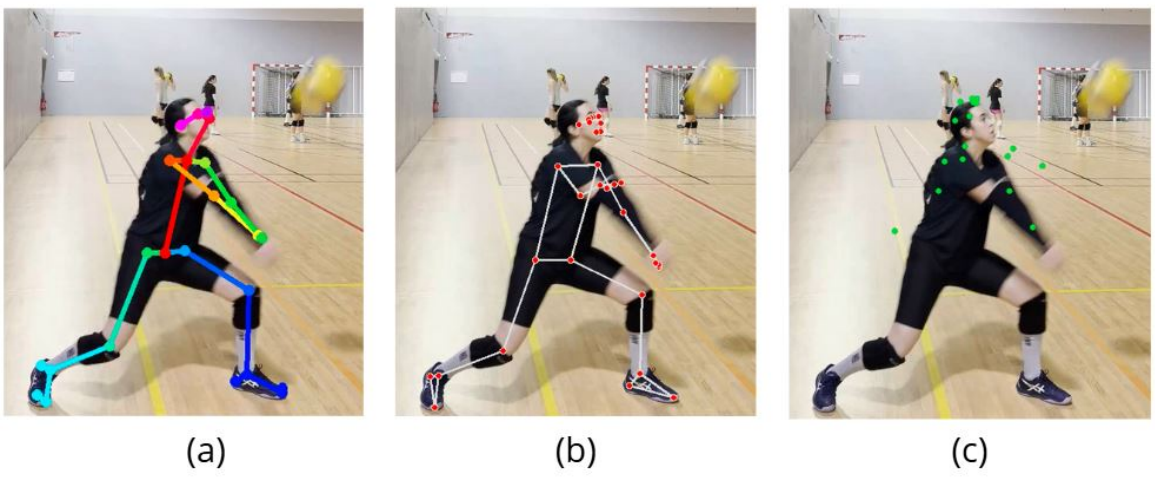

Figure 13 – Réception — comparaison visuelle sur la même frame annotée par OpenPose (a), BlazePose (b) et MoveNet (c). Contexte CPU ; seuil $\theta = 0{,}2$ pour BlazePose/MoveNet.

4.1.3 Attaque (smash)

Outil	Frames	Temps moyen (ms)	Écart-type (ms)	Taux détection (%)
OpenPose (CPU)	129	83 314,31	*2 499*	100,0
BlazePose	129	97,36	66,41	41,9
MoveNet (Thunder)	129	69,84	56,09	100,0

Table 4.3 – Attaque : métriques par outil (rapports de scripts).

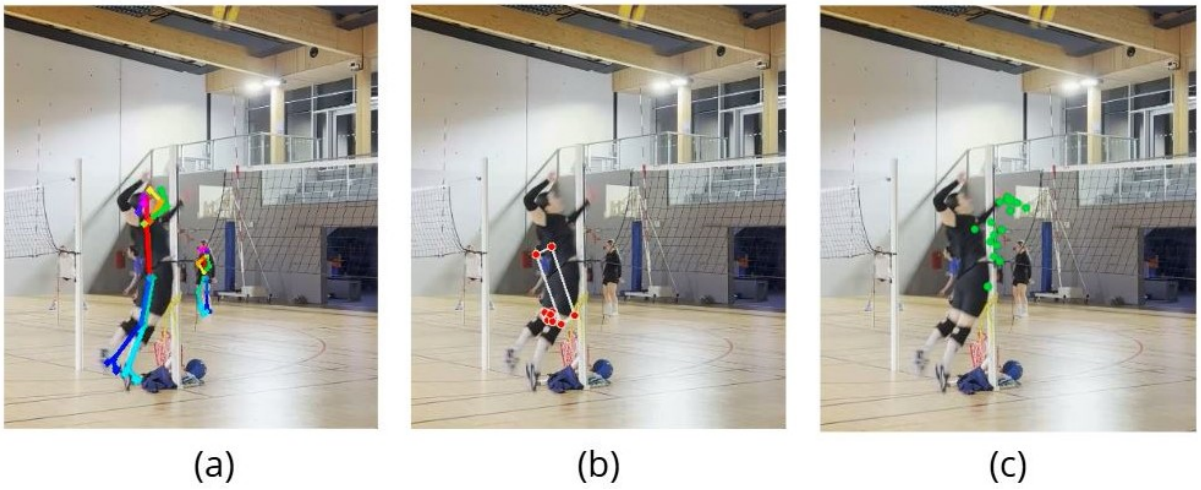

Figure 14 – Attaque — comparaison visuelle sur la même frame annotée par OpenPose (a), BlazePose (b) et MoveNet (c). Contexte CPU ; seuil $\theta = 0{,}2$ pour BlazePose/MoveNet.

Remarques (i) La chute du taux de détection BlazePose en *attaque* (41,9%) reflète des pertes fréquentes sur les extrémités (mains/poignets) lors de mouvements explosifs. (ii) **MoveNet** présente des temps **cohérents** selon les gestes (61–70 ms/frame) avec une variation modérée (48–56 ms), ce qui confirme son intérêt pour une analyse pratique en contexte CPU. (iii) **OpenPose** atteint 100% de détection mais ses temps par frame (∼83 s) rendent l'usage CPU impraticable pour une analyse fluide.

4.2 Analyse comparative

4.2.1 OpenPose

Architecture *bottom-up* à *Part Affinity Fields*, OpenPose offre une **précision visuelle élevée** sur la structure globale et les membres supérieurs, utile pour des analyses fines du geste (angles coude/épaule au service, orientation du tronc). Dans nos conditions **CPU-only**, son coût computationnel **très élevé** (∼83 s/frame) le disqualifie pour une exploitation opérationnelle. L'intérêt d'OpenPose réapparaît avec **accélération GPU** en **multi-personnes**, où sa robustesse d'assemblage des keypoints est un avantage structurel.

4.2.2 BlazePose

Optimisé mobile (approche en deux étages), BlazePose présente un **bon débit** (∼100–170 ms/frame) et une **stabilité** correcte en *réception* (99% de détection).

En *service* (69,9%) et surtout en *attaque* (41,9%), on observe des **pertes d'extrémités** (mains/pieds) et un squelette parfois **instable** sous vitesses angulaires élevées et occlusions (avant-bras proches du tronc). Ces limites sont cohérentes avec une optimisation orientée *fitness/well-being* et des scènes moins extrêmes que le smash.

4.2.3 MoveNet (Thunder)

Bon compromis précision/latence sur *service* et *réception* (environ 62 ms/frame, 100% de détection) et *attaque* (70 ms/frame, 100%). Les **extrémités** peuvent rester **moins fiables** (mains/pieds parfois fluctuants), ce qui impacte des mesures fines (orientation des mains en manchette, par exemple). Pour de l'inférence pure encore plus stable, on pourrait relancer sans écriture vidéo et activer XLA/*pre-warm*.

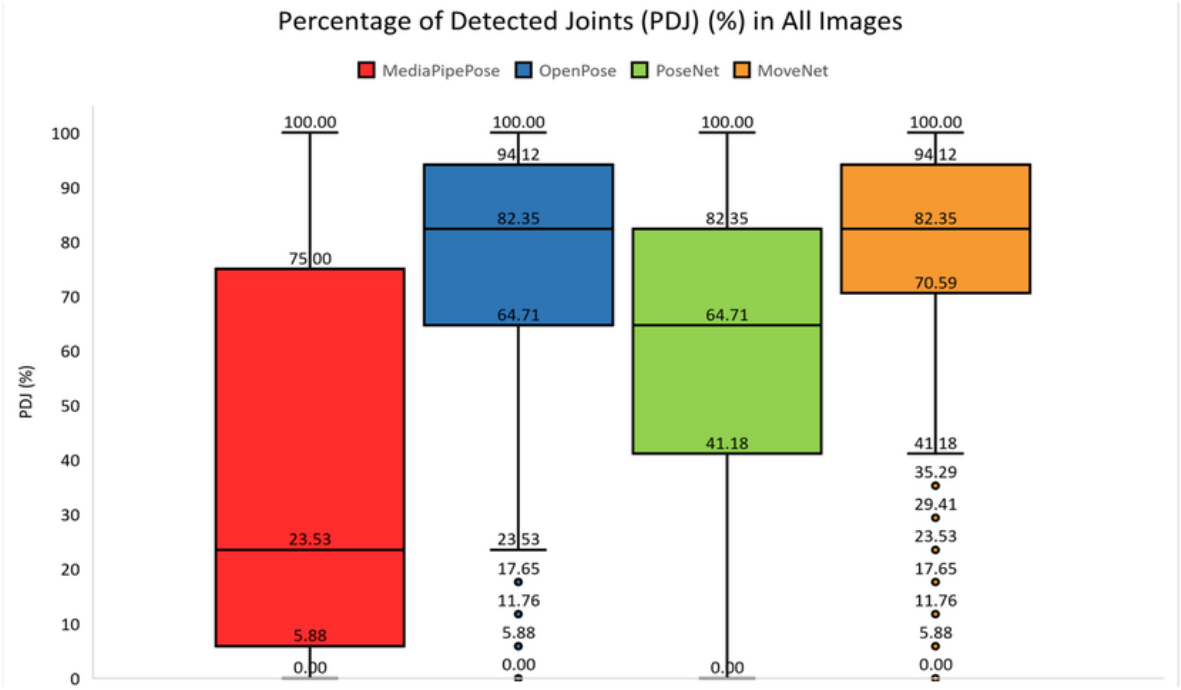

Figure 15 – Comparaison des performances (PDJ – Percentage of Detected Joints) entre différents modèles d'estimation de pose humaine.

4.2.4 Comparaison transversale et adéquation au volleyball

- **Précision visuelle globale** : OpenPose $\geq$ MoveNet $>$ BlazePose (notamment sur les membres supérieurs), au prix d'un **coût** prohibitif en CPU.
- **Latence CPU** : MoveNet $\approx$ 62–70 ms $<$ BlazePose (100–170 ms) $\ll$ OpenPose ($\sim$83 s).
- **Robustesse aux mouvements explosifs** : MoveNet et OpenPose gardent la **detection** (100%), BlazePose décroche en *attaque*.
- **Extrémités (mains/pieds)** : OpenPose $\geq$ MoveNet $>$ BlazePose. Critique pour la *réception* (manchette) et le *service* (orientation de la main).

Réponse à la problématique (contexte CPU) : pour une analyse *opérationnelle* au gymnase (caméra fixe, usage quasi temps réel), **MoveNet Thunder** ressort comme **choix par défaut** (débit/stabilité) ;

BlazePose peut compléter pour le tronc/posture mais reste **fragile** en attaque ; **OpenPose** n'est exploitable que sous **GPU** ou hors-ligne.

4.3 Limites et perspectives

Limites.

- **Absence de vérité terrain annotée** : la « précision » est **qualitative/semi-quantitative**. Des métriques objectives (PCK, mAP) exigeraient un *dataset* annoté.
- **CPU-only** : pénalise fortement OpenPose et ne reflète pas ses performances optimales (GPU).
- **Protocole restreint** : 1 caméra fixe, 1 joueur, 1 environnement intérieur ; généralisation limitée (occlusions, angles variés, arrière-plans changeants, multi-personnes).
- **Non-comparabilité parfaite** des taux de détection : définition différente pour OpenPose (présence `people[]`) vs. seuil $\theta = 0{,}2$ & règle « ≥ 5 points » pour BlazePose/MoveNet.

Perspectives.

- **Évaluation objective** : constituer un sous-ensemble annoté (frames clés) pour calculer PCK/-mAP ; ou réutiliser des datasets publics proches et filmer des *poses étalons*.
- **Accélération** : exécuter OpenPose sur **GPU** et re-mesurer ; *pre-warm*/XLA pour MoveNet ; mesurer *inférence pure* sans écriture vidéo.
- **Diversité des conditions** : angles multiples, plusieurs joueurs, éclairages variés ; tests en **multi-personnes** (force d'OpenPose).
- **Hybrides et post-traitements** : fusion MoveNet (débit) + heuristiques d'extrémités ; lissage temporel ; détection des phases du geste ; fine-tuning sur données de volleyball.

4.4 Conclusion du chapitre

Dans le **contexte CPU** retenu, **MoveNet Thunder** offre le **meilleur compromis** pour une analyse pratique des gestes de volleyball (62–70 ms/frame selon les gestes, 100% de détection), malgré des pertes possibles sur les **extrémités**. **BlazePose** est exploitable pour la posture globale mais se **dégrade** sur l'attaque (41,9% de détection). **OpenPose** fournit la **meilleure précision visuelle** mais est **impraticable** en CPU ($\sim$83 s/frame) ; son intérêt réapparaîtra avec **GPU** et en **multi-personnes**. Ces constats guident la conclusion générale : pour un usage *terrain* orienté performance/ergonomie, MoveNet est le **candidat principal**, complété par des post-traitements ou des approches hybrides ; pour une *analyse fine* des membres supérieurs, OpenPose (sous GPU) reste une référence.

Chapitre 5

Conclusion et perspectives

Ce mémoire s'est inscrit dans une question simple à énoncer mais exigeante à traiter : *quel outil d'estimation de pose humaine privilégier pour analyser efficacement des gestes de volleyball, caractérisés par des phases rapides et parfois explosives, dans un contexte matériel réaliste et contraint ?* Pour y répondre, nous avons d'abord posé un cadre théorique (Chapitre 2) distinguant les familles d'architectures et leurs compromis, puis construit une méthodologie reproductible (Chapitre 3) : même machine *CPU-only*, même procédure de captation, mêmes métriques, et des scripts documentés — fournis intégralement en **Annexe A** — qui transforment les vidéos en sorties annotées et en rapports chiffrés. Le Chapitre 4 a rassemblé et discuté les résultats sur trois gestes emblématiques (*service*, *réception*, *attaque*).

Au terme de cette démarche, une image nette se dessine. **MoveNet Thunder** s'impose comme le meilleur compromis dans le cadre retenu : des temps par image faibles et réguliers (de l'ordre de 62 à 70 ms selon le geste) et un taux de détection maintenu à 100 %, conditions qui rendent l'analyse praticable avec une simple caméra et un CPU. Cette efficacité a toutefois un revers attendu : les extrémités — mains et pieds — peuvent manquer ou fluctuer, ce qui invite à la prudence pour des mesures très fines (orientation de la main en manchette, par exemple). **BlazePose** confirme son positionnement « léger » et robuste pour la posture globale, notamment en *réception* où le taux de détection atteint 99 % ; mais il décroche en *attaque* (41,9 % dans nos tests), là où la vitesse angulaire et les occlusions sollicitent fortement la topologie des membres supérieurs. À l'autre extrême, **OpenPose** demeure la référence visuelle : la structure globale et les segments supérieurs sont finement saisis, et la détection reste complète. En revanche, le coût de calcul en CPU est prohibitif (environ 83 s par image dans nos conditions), ce qui réserve son usage à un traitement hors-ligne ou, plus simplement, à une exécution sous **GPU** où il retrouve sa pertinence — notamment en contexte multi-personnes, pour lequel sa logique d'assemblage des *keypoints* a été conçue.

Ces observations empiriques prolongent directement les propriétés architecturales décrites au Chapitre 2. L'approche *bottom-up* d'OpenPose et ses *Part Affinity Fields* expliquent sa précision et sa robustesse structurelle, mais aussi son coût. La chaîne *deux étages* et l'optimisation mobile de BlazePose justifient sa rapidité, au prix d'une sensibilité accrue aux mouvements extrêmes des membres supérieurs. Le design de MoveNet, pensé pour allier compacité et réactivité, éclaire la régularité observée en CPU, avec les limites attendues sur les extrémités. Autrement dit, la réponse à la problématique ne se réduit pas à « un gagnant universel », mais à un *choix outillé* par le contexte : en simple caméra et CPU, MoveNet est la solution par défaut pour une analyse opérationnelle ; pour une analyse biomécanique plus

fine des membres supérieurs, OpenPose redevient pertinent à condition de disposer d'un GPU ; BlazePose fournit un socle léger et exploitable pour le suivi postural, à manier avec discernement lors des phases les plus rapides.

Comme tout travail appliqué, cette étude a des limites. L'absence de vérité terrain annotée nous a conduits à une évaluation qualitative ou semi-quantitative de la précision, centrée sur le taux de détection et la stabilité temporelle. Le choix du *CPU-only* garantit l'équité de comparaison mais pénalise OpenPose, dont les performances optimales supposent une accélération matérielle. Enfin, le protocole était volontairement restreint (caméra fixe, un joueur, un environnement intérieur), de sorte que la généralisation à d'autres configurations — angles, arrière-plans, multi-personnes — doit être faite avec prudence. Ces contraintes n'affaiblissent pas les conclusions ; elles en définissent le domaine de validité et tracent la feuille de route.

En définitive, ce mémoire ne propose pas seulement un classement ponctuel d'algorithmes : il met à disposition une *méthodologie reproductible*, un jeu d'outils et des scripts, et une grille de lecture pour choisir en connaissance de cause l'estimateur de pose adapté au volleyball et au contexte de déploiement. Cette base peut être réutilisée telle quelle pour élargir l'étude, intégrer de nouvelles architectures ou passer à la 3D. Elle indique surtout qu'un choix raisonné — MoveNet pour l'opérationnel en CPU, OpenPose+GPU pour l'analyse fine, BlazePose pour un suivi léger — est non seulement possible, mais déjà utile à la pratique d'entraînement que vise ce travail.

Bibliographie

[1] Zhe Cao et al. “OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. Dans : *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, n° 1, pp. 172–186, 2021. DOI : 10.1109/TPAMI.2019.2929257.

[2] Valentin Bazarevsky et al. “BlazePose: On-device Real-time Body Pose tracking”. Dans : *arXiv preprint arXiv:2006.10204*. 2020. Disponible à l’URL : https://arxiv.org/abs/2006.10204.

[3] Bin Xiao, Haiping Wu et Yichen Wei. “Simple Baselines for Human Pose Estimation and Tracking”. Dans : *European Conference on Computer Vision (ECCV)*, pp. 466–481, 2018. DOI : 10.1007/978-3-030-01231-1_29.

[4] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. Dans : *European Conference on Computer Vision (ECCV)*, pp. 740–755, 2014. DOI : 10.1007/978-3-319-10602-1_48.

[5] Mykhaylo Andriluka et al. “2D Human Pose Estimation: New Benchmark and State of the Art Analysis”. Dans : *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3686–3693, 2014. DOI : 10.1109/CVPR.2014.471.

[6] Catalin Ionescu et al. “Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments”. Dans : *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, n° 7, pp. 1325–1339, 2014. DOI : 10.1109/TPAMI.2013.248.

[7] Alejandro Newell, Zhiao Huang et Jia Deng. “Associative Embedding: End-to-End Learning for Joint Detection and Grouping”. Dans : *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2277–2287, 2017.

[8] Ke Sun et al. “Deep High-Resolution Representation Learning for Human Pose Estimation”. Dans : *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5693–5703, 2019. DOI : 10.1109/CVPR.2019.00584.

[9] Google Research. “MoveNet: Ultra fast and accurate pose detection model”. Dans : *TensorFlow Blog*. 2021. Disponible à l’URL : https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html.

[10] Yifan Chen et al. “Cascaded pyramid network for multi-person pose estimation”. Dans : pp. 7103–7112, 2018. DOI : 10.1109/CVPR.2018.00742.

[11] Graham Thomas et al. “Computer vision for sports: Current applications and research topics”. Dans : *Computer Vision and Image Understanding*, vol. 159, pp. 3–18, 2017. DOI : 10.1016/j.cviu.2017.04.011.

[12] Steffi L. Colyer et al. “A Review of the Evolution of Vision-Based Motion Analysis and the Integration of Advanced Computer Vision Methods Towards Developing a Markerless System”. Dans : *Sports Medicine - Open*, vol. 4, n° 1, p. 24, 2018. DOI : 10.1186/s40798-018-0139-y.

[13] Yi Yang et Deva Ramanan. “Articulated human detection with flexible mixtures of parts”. Dans : vol. 35. 12, pp. 2878–2890, 2013. doi : 10.1109/TPAMI.2012.261.

[14] Yilun Chen et al. “Cascaded pyramid network for multi-person pose estimation”. Dans : pp. 7103–7112, 2018. doi : 10.1109/CVPR.2018.00742.

[15] Shih-En Wei et al. “Convolutional pose machines”. Dans : *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4724–4732, 2016. doi : 10.1109/CVPR.2016.511.

Glossaire

Backbone Réseau de neurones extracteur de caractéristiques

biomécanique (contexte) Analyse d'angles, vitesses et amplitudes dérivées des keypoints estimés pour interpréter la qualité technique des gestes

champs d'affinité (PAF) Champs de vecteurs d'OpenPose qui encodent la cohérence de liaison entre deux keypoints afin d'assembler le squelette multi-personnes

détection OpenPose Pour OpenPose, une frame est comptée comme détectée si le JSON contient au moins une entrée dans `people[]`

écart-type du temps Variabilité (ms) des temps d'inférence d'une vidéo; utilisé pour caractériser la stabilité temporelle

Flou de mouvement Dégradation de l'image due à la vitesse du mouvement

frame détectée Dans ce mémoire, une image est dite détectée si $\geq$**5 keypoints** ont une confiance $> \theta$; règle appliquée à BlazePose et MoveNet

HeatMap Carte de probabilité spatiale pour localiser les keypoints

keypoint Point anatomique estimé (articulation/extrémité) avec coordonnées et score de confiance

landmark (MediaPipe) Terme employé par MediaPipe/BlazePose pour un keypoint accompagné d'un score de visibilité

Oclusion Masquage partiel ou total d'articulations dans l'image

Single-shot Architecture qui effectue détection et régression en une seule passe

Supervision intermédiaire Entraînement avec perte calculée à chaque étape du réseau multi-stage

taux de détection Proportion (%) de frames détectées sur l'ensemble de la séquence, au sens défini pour les frames détectées ou la détection OpenPose

topologie de keypoints Schéma des points articulaires et de leurs liaisons. Utilisés ici : *COCO-17* (MoveNet), *Body25* (OpenPose) et *BlazePose-33*

Liste des acronymes

CNN Convolutional Neural Network

COCO Common Objects in Context (schéma 17 points)

FPS Frames Per Second

HPE Estimation de pose humaine

mAP mean Average Precision

NMS Non-Maximum Suppression

OKS Object Keypoint Similarity

PAF Part Affinity Fields

PCK Percentage of Correct Keypoints

TF TensorFlow

TF Hub TensorFlow Hub

VGG Visual Geometry Group

XLA Accelerated Linear Algebra

Annexe A

Scripts Python complets

Cette annexe rassemble l'intégralité des scripts utilisés pour exécuter les trois outils HPE (référencés au Chapitre 3, Section 3.4). Les chemins (VIDEO_PATH, etc.) doivent être adaptés à votre arborescence.

Environnement logiciel (versions)

Python 3.10.12 NumPy 1.26.4 OpenCV 4.11.0.86 MediaPipe 0.10.21
TensorFlow (`tf.__version__` à l'exécution) TensorFlow Hub (`hub.__version__`, si disponible)

A.1 Script OpenPose

```
import os
import subprocess
import json
import time
import numpy as np
from datetime import datetime

VIDEO_PATH = "../memoire_pose/service.mp4"
OPENPOSE_BIN = "./build/examples/openpose/openpose.bin"
OUTPUT_JSON_DIR = "output_openpose_json"
OUTPUT_VIDEO = "output_openpose_video.mp4"
MODEL_FOLDER = "models"
REPORT_DIR = " resultats_tests "

os.makedirs(OUTPUT_JSON_DIR, exist_ok=True)
os.makedirs(REPORT_DIR, exist_ok=True)
for f in os.listdir(OUTPUT_JSON_DIR):
    os.remove(os.path.join(OUTPUT_JSON_DIR, f))
if os.path.exists(OUTPUT_VIDEO): os.remove(OUTPUT_VIDEO)

print("[INFO] Starting OpenPose analysis ... ")
start_time = time.time()
res = subprocess.run([
```

```
        OPENPOSE_BIN, "--video", VIDEO_PATH,
        "--write_json", OUTPUT_JSON_DIR,
        "--display", "0", "--render_pose", "1",
        "--write_video", OUTPUT_VIDEO, "--model_folder", MODEL_FOLDER
    ], check=False)
    end_time = time.time()
    print("[INFO] OpenPose analysis completed.")

    json_files = sorted(f for f in os.listdir(OUTPUT_JSON_DIR) if f.endswith(".json"))
    detection_flags, json_parse_times = [], []
    for filename in json_files:
        json_path = os.path.join(OUTPUT_JSON_DIR, filename)
        t1 = time.time()
        with open(json_path, 'r') as f:
            data = json.load(f)
        t2 = time.time()
        json_parse_times.append(t2 - t1)
        detection_flags.append(1 if data.get("people") else 0)

    frames_processed = len(json_files)
    avg_json_parse_time = np.mean(json_parse_times) * 1000 if json_parse_times else 0
    std_json_parse_time = np.std(json_parse_times) * 1000 if json_parse_times else 0
    detection_rate = np.mean(detection_flags) * 100 if detection_flags else 0

    total_runtime = end_time - start_time
    real_avg_time = (total_runtime / frames_processed) * 1000 if frames_processed else 0

    print("\n=== OPENPOSE RUNTIME REPORT ===")
    print(f"Model: OpenPose (CPU)")
    print(f"Video: {VIDEO_PATH}")
    print(f"Output video: {OUTPUT_VIDEO}")
    print(f"Frames processed: {frames_processed}")
    print(f"Real average time per frame (total run): {real_avg_time:.2f} ms")
    print(f"JSON parse avg time: {avg_json_parse_time:.2f} ms")
    print(f"Std deviation of JSON parse time: {std_json_parse_time:.2f} ms")
    print(f"Detection rate: {detection_rate:.1f}%")

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    report_path = os.path.join(REPORT_DIR, f"rapport_openpose_{timestamp}.txt")
    with open(report_path, "w") as f:
        f.write("=== OPENPOSE RUNTIME REPORT ===\n")
        f.write(f"Model: OpenPose (CPU)\n")
        f.write(f"Video: {VIDEO_PATH}\n")
        f.write(f"Output video: {OUTPUT_VIDEO}\n")
        f.write(f"Frames processed: {frames_processed}\n")
        f.write(f"Real average time per frame (total run): {real_avg_time:.2f} ms\n")
        f.write(f"JSON parse avg time: {avg_json_parse_time:.2f} ms\n")
```

```
        f.write(f"Std deviation of JSON parse time: {std_json_parse_time:.2f} ms\n")
        f.write(f"Detection rate : { detection_rate :.1f}%\n")
print(f"[INFO] Report saved: {report_path}")
```

A.2 Script BlazePose

```
import cv2, mediapipe as mp, time, numpy as np

VIDEO_PATH = "reception.mp4"
OUTPUT_VIDEO = "output_blazepose_video.mp4"
REPORT_FILE = "report_blazepose.txt"
THRESHOLD = 0.2
MODEL_NAME = "BlazePose"

mp_pose = mp.solutions.pose
pose = mp_pose.Pose(static_image_mode=False, min_detection_confidence = 0.5,
    model_complexity=2)
mp_drawing = mp.solutions.drawing_utils

cap = cv2.VideoCapture(VIDEO_PATH)
if not cap.isOpened(): print(f"[ERREUR] Impossible d'ouvrir la vidéo : {VIDEO_PATH}");
    exit(1)

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)); height = int(cap.get(cv2.
    CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(OUTPUT_VIDEO, fourcc, fps, (width, height))

frame_count, detected_frames = 0, 0
inference_times = []
print("[INFO] Traitement de la vidéo ...")

while True:
    ret, frame = cap.read()
    if not ret: break
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    start = time.time(); results = pose.process(frame_rgb); end = time.time()
    inference_times.append((end - start) * 1000); frame_count += 1

    if results.pose_landmarks:
        detected = sum(1 for lm in results.pose_landmarks.landmark if lm.visibility >
    THRESHOLD)
        if detected >= 5: detected_frames += 1
        mp_drawing.draw_landmarks(frame, results.pose_landmarks, mp_pose.
    POSE_CONNECTIONS)
```

```
        out.write(frame)

cap.release(); out.release(); pose.close()

avg_time = np.mean(inference_times); std_time = np.std(inference_times)
detection_rate = (detected_frames / frame_count) * 100

print(f"\n=== {MODEL_NAME.upper()} RUNTIME REPORT ===")
print(f"Model: {MODEL_NAME}")
print(f"Video: {VIDEO_PATH}")
print(f"Output video: {OUTPUT_VIDEO}")
print(f"Frames processed: {frame_count}")
print(f"Average time per frame: {avg_time:.2f} ms")
print(f"Std deviation of time: {std_time:.2f} ms")
print(f"Detection rate ( {THRESHOLD}): {detection_rate:.1f}%")

with open(REPORT_FILE, "w") as f:
    f.write(f"=== {MODEL_NAME.upper()} RUNTIME REPORT ===\n")
    f.write(f"Model: {MODEL_NAME}\n")
    f.write(f"Video: {VIDEO_PATH}\n")
    f.write(f"Output video: {OUTPUT_VIDEO}\n")
    f.write(f"Frames processed: {frame_count}\n")
    f.write(f"Average time per frame: {avg_time:.2f} ms\n")
    f.write(f"Std deviation of time: {std_time:.2f} ms\n")
    f.write(f"Detection rate ( {THRESHOLD}): {detection_rate:.1f}%\n")
```

A.3 Script MoveNet (Thunder)

```
import tensorflow as tf, tensorflow_hub as hub
import numpy as np, cv2, time

VIDEO_PATH = "attaque.mp4"
OUTPUT_VIDEO = "output_movenet_attaque_video.mp4"
REPORT_FILE = "report_movenet_attaque.txt"
MODEL_URL = "https://tfhub.dev/google/movenet/singlepose/thunder/4"
THRESHOLD = 0.2
MODEL_NAME = "MoveNet Thunder"

model = hub.load(MODEL_URL); movenet = model.signatures['serving_default']

cap = cv2.VideoCapture(VIDEO_PATH)
if not cap.isOpened(): print(f"[ERREUR] Impossible d'ouvrir la vidéo : {VIDEO_PATH}");
    exit(1)

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)); height = int(cap.get(cv2.
    CAP_PROP_FRAME_HEIGHT))
```

```
fps = cap.get(cv2.CAP_PROP_FPS)
fourcc = cv2.VideoWriter_fourcc(*'mp4v'); out = cv2.VideoWriter(OUTPUT_VIDEO, fourcc,
    fps, (width, height))

frame_count, detected_frames = 0, 0
inference_times = []
print("[INFO] Traitement de la vidéo ...")

while True:
    ret, frame = cap.read()
    if not ret: break
    img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    img_resized = tf.image.resize_with_pad(tf.convert_to_tensor(img_rgb), 256, 256)
    input_tensor = tf.expand_dims(tf.cast(img_resized, dtype=tf.int32), axis=0)

    start = time.time(); outputs = movenet(input_tensor); end = time.time()
    inference_times.append((end - start) * 1000); frame_count += 1

    keypoints = outputs['output_0'].numpy()[0, 0, :, :]
    num_detected = np.sum(keypoints[:, 2] > THRESHOLD)
    if num_detected >= 5: detected_frames += 1

    for y, x, conf in keypoints:
        if conf > THRESHOLD:
            cx, cy = int(x * width), int(y * height)
            cv2.circle(frame, (cx, cy), 4, (0, 255, 0), -1)
    out.write(frame)

cap.release(); out.release()

avg_time = np.mean(inference_times); std_time = np.std(inference_times)
detection_rate = (detected_frames / frame_count) * 100

print(f"\n=== {MODEL_NAME.upper()} RUNTIME REPORT ===")
print(f"Model: {MODEL_NAME}")
print(f"Video: {VIDEO_PATH}")
print(f"Output video: {OUTPUT_VIDEO}")
print(f"Frames processed: {frame_count}")
print(f"Average time per frame: {avg_time:.2f} ms")
print(f"Std deviation of time: {std_time:.2f} ms")
print(f"Detection rate ( {THRESHOLD}): {detection_rate:.1f}%")

with open(REPORT_FILE, "w") as f:
    f.write(f"=== {MODEL_NAME.upper()} RUNTIME REPORT ===\n")
    f.write(f"Model: {MODEL_NAME}\n")
    f.write(f"Video: {VIDEO_PATH}\n")
    f.write(f"Output video: {OUTPUT_VIDEO}\n")
```

```
63        f.write(f"Frames processed: {frame_count}\n")
64        f.write(f"Average time per frame: {avg_time:.2f} ms\n")
65        f.write(f"Std deviation of time: {std_time:.2f} ms\n")
66        f.write(f"Detection rate ( {THRESHOLD}): {detection_rate:.1f}%\n")
```

Annexe B — Extraits de logs et sorties console

```
anais@memoire:~/openpose$ python3 complete_openpose.py
[INFO] Starting OpenPose analysis...
Starting OpenPose demo...
Configuring OpenPose...
Starting thread(s)...
---------------------------------- WARNING ----------------------------------
We have introduced an additional boost in accuracy in the CUDA version of about 0.2% with respect to the CPU/OpenCL versions. We will
 not port this to CPU given the considerable slow down in speed it would add to it. Nevertheless, this accuracy boost is almost insig
nificant so the CPU/OpenCL versions can be safely used.
------------------------------- END WARNING ---------------------------------
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg developers
built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/i
nclude/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-gnutls --enable-ladspa --enable-libaom --enable-libass
 --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libcodec2 --enable-libdav1d --enable-libflite --enabl
e-libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --enable-libjack --enable-libmp3lame --enabl
e-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librabbitmq --enable-librubberband -
-enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libsrt --enable-libssh --enable-libtheora --enable-li
btwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libxvid -
-enable-libzimg --enable-libzmq --enable-libzvbi --enable-lv2 --enable-omx --enable-openal --enable-opencl --enable-opengl --enable-s
dl2 --enable-pocketsphinx --enable-librsvg --enable-libmfx --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-chromapri
nt --enable-frei0r --enable-libx264 --enable-shared
libavutil      56. 70.100 / 56. 70.100
libavcodec     58.134.100 / 58.134.100
libavformat    58. 76.100 / 58. 76.100
libavdevice    58. 13.100 / 58. 13.100
libavfilter     7.110.100 /  7.110.100
libswscale      5.  9.100 /  5.  9.100
libswresample   3.  9.100 /  3.  9.100
libpostproc    55.  9.100 / 55.  9.100
Temporarily saving video frames as JPG images in: output_openpose_video_r8904530ijyiopf9034jiop4g90j0yh795640h38j
```

Figure 16 – OpenPose (CPU) — lancement de l'analyse et initialisation FFmpeg.

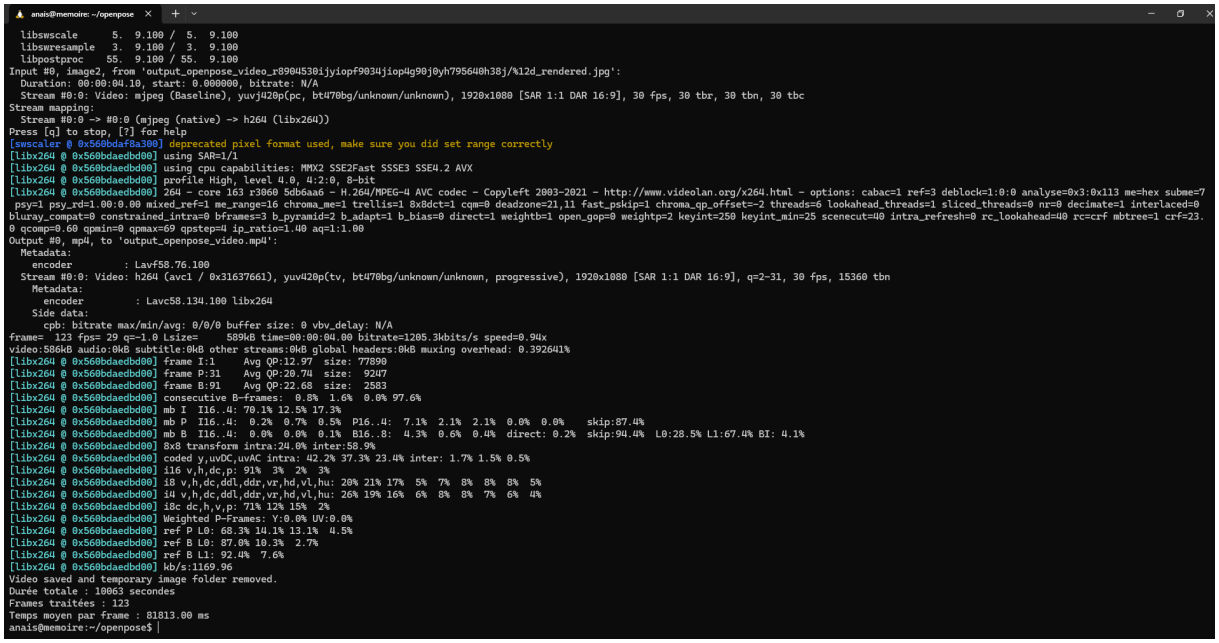

Figure 17 – OpenPose (CPU) — encodage FFmpeg et statistiques finales (durée totale, frames traitées, temps moyen par frame).

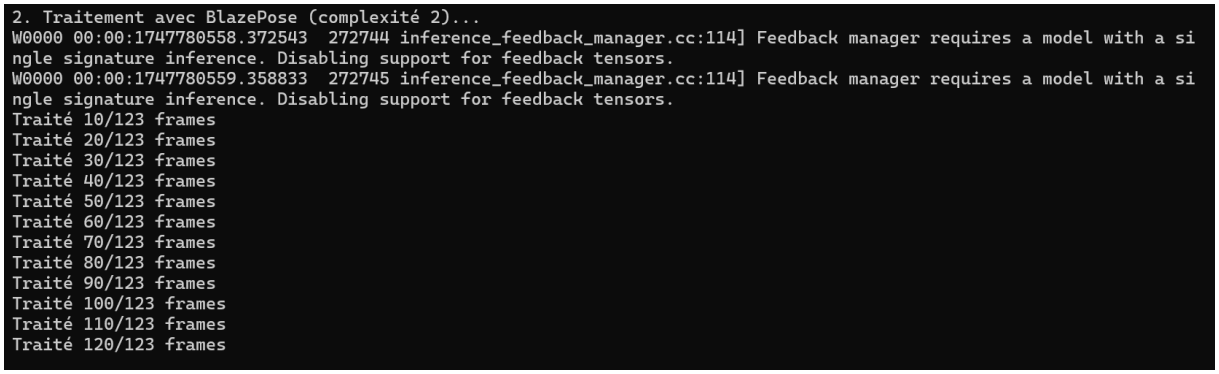

Figure 18 – BlazePose — progression du traitement (« Traité $X/123$ frames ») et messages d'avertissement standards.

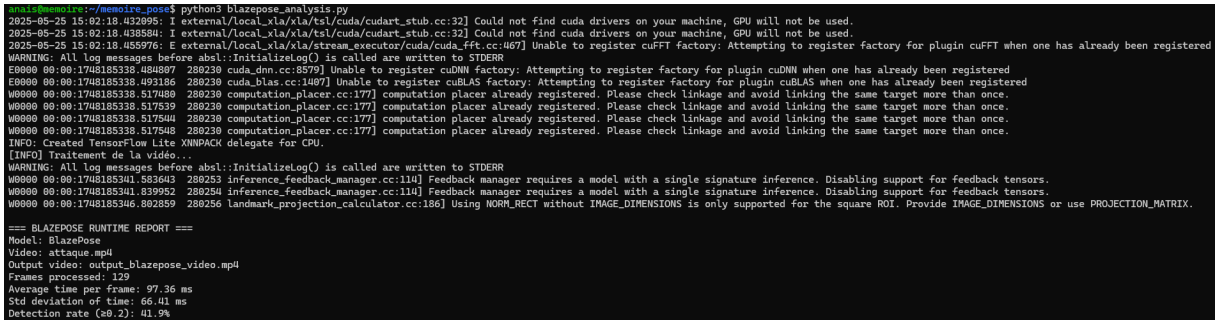

Figure 19 – BlazePose — *Runtime Report* (ex. `attaque.mp4`) : nombre de frames, temps moyen, écart-type et taux de détection.

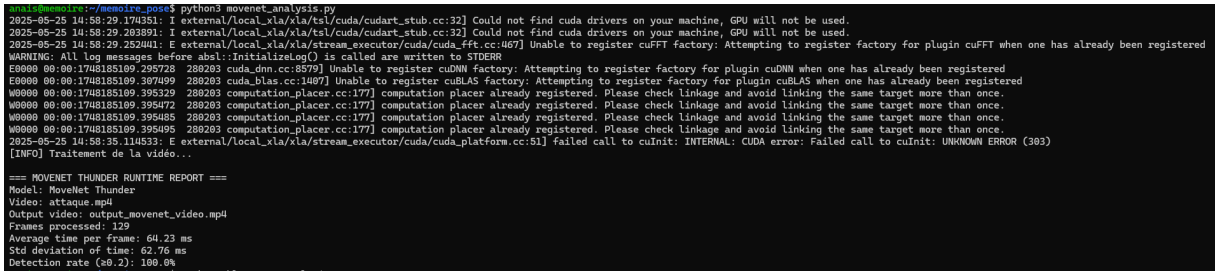

FIGURE 20 – MoveNet Thunder — *Runtime Report* (ex. `service.mp4`) : nombre de frames, temps moyen, écart-type et taux de détection.

Résumé : Ce mémoire étudie le choix d'un estimateur de pose humaine (HPE) pour l'analyse de gestes de volleyball dans un contexte réaliste et contraint (caméra fixe, exécution *CPU-only*). Après une synthèse des familles d'architectures (OpenPose, BlazePose, MoveNet), nous proposons une méthodologie reproductible : captation de séquences réelles (*service*, *réception*, *attaque*), scripts d'inférence et de mesure, sorties annotées (vidéo, JSON) et rapports chiffrés. Les résultats montrent que **MoveNet Thunder** offre le meilleur compromis en CPU (environ 62–70 ms/image selon le geste, détection maintenue à 100 %), rendant l'analyse praticable au gymnase ; ses limites portent sur les extrémités (mains/pieds) pour des mesures très fines. **BlazePose** est rapide et stable pour la posture globale, mais décroche nettement sur l'attaque (41,9 % de détection dans nos tests). **OpenPose** fournit la meilleure fidélité visuelle (100 % de détection) au prix d'un coût prohibitif en CPU (environ 83 s/image) et retrouve sa pertinence sous GPU et en multi-personnes.

Nous discutons les limites (absence de vérité terrain annotée, protocole à caméra unique) et ouvrons des perspectives concrètes : constitution d'un sous-ensemble annoté pour des métriques objectives (PCK/mAP), mesures sous GPU en inférence pure, diversification des conditions de prise de vue et post-traitements pour renforcer les extrémités ou segmenter automatiquement les phases du geste.

Mots clefs : Estimation de pose humaine (HPE), volleyball, analyse sportive, vision par ordinateur, biomécanique, mouvements explosifs, performance, temps réel, CPU, précision, OpenPose, BlazePose, MoveNet.

Abstract : This thesis investigates which human pose estimation (HPE) model is best suited to analyze volleyball techniques under realistic constraints (single fixed camera, *CPU-only* execution). After reviewing major architectures (OpenPose, BlazePose, MoveNet), we present a reproducible pipeline : real video capture (serve, reception, attack), inference and measurement scripts, annotated outputs (video, JSON) and numeric reports. Results show that **MoveNet Thunder** provides the best CPU trade-off (about 62–70 ms/frame depending on the skill, with 100 % detection), enabling practical on-court analysis ; its main limitation concerns extremities (hands/feet) for very fine measurements. **BlazePose** is fast and stable for global posture but drops on attacks (41.9 % detection in our tests). **OpenPose** delivers the highest visual fidelity (100 % detection) at a prohibitive CPU cost (around 83 s/frame), regaining relevance on GPU and in multi-person scenarios.

We discuss limitations (no ground-truth annotations, single-camera protocol) and outline actionable perspectives : building an annotated subset for objective metrics (PCK/mAP), GPU measurements with pure inference, more diverse recording conditions, and post-processing to strengthen extremities and automatically segment motion phases.

Keywords : Human pose estimation (HPE), volleyball, sports analytics, computer vision, biomechanics, explosive movements, performance, real-time, CPU, accuracy, OpenPose, BlazePose, MoveNet.