

Robot suiveur de personnes

Evan Galli, Jilian Lubrat, Eliot Menoret,
Antoine-Marie Michelozzi

- *Optimisation de modèle IA* -

Un robot autonome capable de reconnaître et de suivre des personnes, reposant sur la reconnaissance de gestes prédefinis.

[Repo du Projet](#) — [Repo du Benchmark](#)

2026-01-22

Table des matières

I.	Contexte	3
II.	Critères de benchmark	4
II.1.	Performance temporelle	4
II.2.	Efficacité énergétique	4
II.3.	Qualité de prédiction	4
III.	Plateformes visées	6
III.1.	Raspberry Pi 4 modèle B	6
III.2.	Caméra Luxonis OAK-D Pro V2	6
III.3.	Ordinateur portable	7
IV.	Protocole du benchmark	8
IV.1.	Objectif et conditions expérimentales	8
IV.2.	Architecture de la pipeline d'inférence	8
IV.3.	Mesure de la consommation	9
IV.4.	Scénarios	9
IV.4.1.	Scénario 1: Caméra OAK-D Pro	9
IV.4.2.	Scénario 2 : Raspberry Pi 4	10
IV.4.3.	Scénario 3 : Ordinateur portable	10
V.	Les modèles et les optimisations	11
V.1.	Optimisations générales	11
V.1.1.	Quantification	11
V.1.2.	Pruning structuré	12
V.2.	Optimisations spécifiques aux plateformes	13
VI.	Benchmark	14
VII.	Conclusion	17

Avant-propos

Des outils d'intelligence artificielle ont été utilisés pour l'amélioration de la rédaction de ce rapport.

I. Contexte

Le projet global vise à concevoir un robot capable de suivre une personne et de reconnaître des gestes afin de contrôler ses déplacements (le stopper, lui dire de reprendre le suivi, etc.). Pour répondre à ces besoins, il est nécessaire de disposer d'un modèle de vision capable de détecter les personnes en temps réel sur un système embarqué.

Dans ce contexte, nous avons sélectionné le modèle [YOLOv11 nano](#)° avec segmentation d'instances. Ce choix est motivé par la nécessité d'estimer la distance entre la personne et le robot. Cela nécessite de segmenter l'instance de la personne sur l'image de profondeur afin de réaliser ce calcul. De plus, ce choix repose sur son architecture améliorée qui offre une efficacité supérieure à celle de ses prédecesseurs.

Comparativement à YOLOv8, YOLOv11 réduit le nombre de paramètres d'environ 22%, allégeant la charge sur notre système embarqué tout en augmentant la précision (mAP). Face à YOLOv10, nous avons privilégié la v11 pour son support natif de la segmentation d'instance, élément critique pour notre calcul de profondeur. Cette précision accrue nous permet de fiabiliser l'estimation de la distance personne-robot sans compromettre la fluidité de la navigation en temps réel.

	YOLOv8	YOLOv10	YOLOv11
Tâches	Segmentation, détection, ...	Détection	Segmentation, détection, ...
Temps d'inference CPU ONNX (ms)	96.1	-	65.8
Nombre de paramètres (M)	3.4	2.3	2.9
mAP Box 50%-95% (%)	36.7	39.5	38.9
mAP Mask 50%-95% (%)	30.5	-	32.0

Tableau 1. – Différences entre les versions de YOLO

L'objectif spécifique de ce rapport est de comparer différentes méthodes d'optimisation du modèle adaptées aux contraintes de l'edge computing, tout en quantifiant l'impact de l'accélération matérielle (NPU/VPU) face à un CPU généraliste.

Cette étude s'appuie sur la mise en place d'une pipeline de benchmarking permettant d'évaluer ces approches selon trois axes principaux, afin de dégager des compromis pertinents entre temps d'inférence, performances et consommation de ressources

II. Critères de benchmark

Afin de qualifier la pertinence des différentes architectures matérielles et logicielles pour notre système de suivi, nous avons défini trois axes d'évaluation distincts. Ces critères visent à garantir que la solution retenue respecte les contraintes de temps réel, d'autonomie énergétique et de fiabilité de détection.

II.1. Performance temporelle

Dans notre cas d'usage, l'objectif est de minimiser la latence pour rester synchronisé avec la réalité, tout en assurant une fréquence (FPS) suffisante pour un suivi fluide.

Pour évaluer cette rapidité, nous nous basons sur le temps d'inférence. Celui-ci correspond au délai introduit spécifiquement par le réseau de neurones, calculé en mesurant la durée écoulée entre l'entrée et la sortie du modèle.

II.2. Efficacité énergétique

Dans un environnement embarqué contraint par l'autonomie de la batterie, l'optimisation énergétique est primordiale. Notre objectif est d'isoler la consommation dynamique imputable au traitement IA, indépendamment de la consommation statique du matériel. Le protocole de mesure consiste à évaluer le différentiel entre la consommation du système en charge (durant l'inférence) et sa consommation basale (au repos/idle).

II.3. Qualité de prédiction

Enfin, l'optimisation de la vitesse et de la consommation ne doit pas se faire au détriment de la perception du robot. L'objectif est de maximiser la précision pour garantir que les personnes sont correctement détectées et segmentées.

La fiabilité du modèle est quantifiée par la mAP (mean Average Precision), calculée selon le standard COCO. Cette métrique se construit en trois étapes :

D'abord, pour qualifier une prédiction, il faut mesurer sa cohérence géométrique avec la réalité. Nous utilisons l'IoU (Intersection over Union), qui représente le rapport entre la surface commune (intersection) et la surface totale (union) des deux zones (prédiction et vérité terrain).

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (1)$$

Un seuil d'IoU est défini pour classer les détections :

- Vrai Positif (TP) : L'IoU dépasse le seuil (détexion correcte).
- Faux Positif (FP) : L'IoU est sous le seuil (mauvaise localisation ou détection d'un objet inexistant).
- Faux Négatif (FN) : Le modèle a manqué une personne présente.

Ensuite, deux indicateurs sont calculés :

- La Précision (P) : La fiabilité des détections positives. Précision = $\frac{TP}{TP+FP}$
- Le Rappel (R) : La capacité à trouver toutes les occurrences. Rappel = $\frac{TP}{TP+FN}$

Enfin, pour synthétiser le compromis entre ces deux valeurs, nous calculons l'Average Precision (AP), l'aire sous la courbe Précision-Rappel.

Cependant, un seuil d'IoU unique (ex: 0.5) est insuffisant pour la précision requise par un robot suiveur. Conformément au protocole COCO, la mAP finale est la moyenne des AP calculées pour 10 seuils d'IoU différents (de 0.5 à 0.95 par pas de 0.05).

$$mAP = \frac{1}{10} \sum_{IoU=0.5}^{0.95} AP_{IoU} \quad (2)$$

Cette méthode pénalise les modèles qui détectent la bonne classe (personne) mais avec un positionnement approximatif (IoU faible).

Le modèle utilisé étant de type YOLOv11n-seg, l'évaluation se fait sur deux niveaux distincts:

- mAP Box : Évalue la précision des cadres englobants (bounding boxes). C'est un indicateur de la capacité du robot à localiser grossièrement la cible.
- mAP Mask : Évalue la précision de la segmentation pixel par pixel (le masque de la silhouette).

Nos résultats présenteront ces deux valeurs afin de vérifier si les optimisations (comme la quantification ou le pruning) dégradent la finesse du masque ou la détection globale (Box).

III. Plateformes visées

Pour valider la portabilité et l'efficacité du modèle de vision, l'évaluation expérimentale s'appuie sur trois plateformes aux profils matériels distincts. L'objectif est de balayer un large spectre de puissance de calcul, allant d'un système embarqué fortement constraint (Raspberry Pi) à un environnement de référence non limité (Ordinateur portable) mais aussi une caméra 3D (Luxonis OAK-D Pro) pouvant supporter des réseaux de neurones. Cette segmentation nous permettra non seulement de mesurer les temps d'inférence bruts, mais aussi d'analyser l'impact de l'accélération matérielle et des optimisations logicielles sur la consommation énergétique globale.

III.1. Raspberry Pi 4 modèle B

Le Raspberry Pi 4 modèle B constitue la plateforme embarqué à ressources limitées de notre étude. Il est équipé d'un processeur ARM Cortex-A72 quad cœur cadencé à 1,5 GHz et ne dispose pas d'accélération matérielle dédiée pour l'inférence de réseaux de neurones.

Cette plateforme représente un cas d'usage réaliste pour un robot mobile fonctionnant sur batterie, où l'inférence est effectuée exclusivement sur CPU. Elle permet d'évaluer la faisabilité du modèle dans un environnement fortement constraint, ainsi que l'impact des optimisations logicielles (quantification, pruning) sur les performances temporelles et énergétiques.

III.2. Caméra Luxonis OAK-D Pro V2

La Luxonis OAK-D Pro V2 est une caméra 3D intégrant un VPU Intel Myriad X, capable de délivrer jusqu'à 4 TOPS (*Trillions Operations Per Second*) avec ses 16 cœurs SHAVEs (*Streaming Hybrid Architecture Vector Engine*) cadencés à environ 700 MHz. À la différence du Raspberry Pi, le calcul d'inférence est ici délesté directement sur le matériel de la caméra.

Cette architecture permet de tirer parti d'une accélération matérielle dédiée tout en libérant les ressources du système hôte. L'OAK-D Pro sert de point de comparaison pour illustrer la plus-value d'un accélérateur spécialisé face à une exécution CPU conventionnelle.

III.3. Ordinateur portable

La plateforme de référence haute performance retenue pour cette étude est un ordinateur portable. Sa configuration matérielle s'articule autour d'un processeur AMD Ryzen 5 8645HS (6 cœurs / 12 threads à 4,3 GHz) et de 32 Go de mémoire vive DDR5. La partie graphique est assurée par une carte NVIDIA RTX 4060 Laptop, dotée de 3072 cœurs CUDA à 2,1 GHz et de 8 Go de VRAM GDDR6.

Les tests y sont conduits selon deux configurations : exécution sur CPU seul et exécution accélérée par GPU (avec basculement CPU si nécessaire). Cette plateforme offre des conditions d'exécution nettement moins restrictives, bénéficiant d'une puissance de calcul supérieure à celle des systèmes embarqués.

Cet environnement sert principalement d'étalement (baseline) pour :

- Comparer les temps d'inférence face aux matériels embarqués.
- Analyser les écarts de consommation énergétique.
- Isoler l'impact des optimisations du modèle dans un contexte non contraint.

Enfin, cette plateforme permet de valider l'intégrité du pipeline d'inférence, garantissant que les limitations observées sur les autres supports découlent bien des contraintes matérielles et non de l'architecture du modèle elle-même.

IV. Protocole du benchmark

IV.1. Objectif et conditions expérimentales

Le protocole de benchmark a pour objectif de mesurer de manière quantitative le temps d'inférence ainsi que la consommation énergétique associés à l'exécution du modèle de segmentation de personnes sur différentes plateformes matérielles. L'ensemble du protocole a été conçu de manière à garantir des conditions d'évaluation identiques et reproductibles, afin de permettre une comparaison équitable entre les différents environnements testés.

Pour chaque appareil, le benchmark exécute l'ensemble des modèles successivement, chaque modèle étant évalué sur un ensemble de 100 images. Les images utilisées sont identiques pour tous les tests et proviennent du dataset COCO, en se limitant à la classe person. Entre l'exécution de deux modèles différents, un temps d'attente de 10 secondes est systématiquement appliqué afin de laisser au système le temps de revenir à un état de consommation au repos. Cette pause permet à la fois de stabiliser le système et de mesurer la consommation énergétique hors inférence, qui servira de référence lors de l'analyse des résultats.

IV.2. Architecture de la pipeline d'inférence

Afin de comparer équitablement les performances de différents environnements d'exécution, une abstraction de l'inférence a été mise en place à travers la classe `InferenceBackend`. Cette classe abstraite définit une interface commune pour l'exécution de l'inférence, indépendamment du matériel ou du framework utilisé. Elle impose notamment une méthode `predict()` que toutes les implémentations concrètes doivent fournir, ce qui permet d'uniformiser la manière dont les modèles sont appelés au sein de la pipeline de benchmark et de rendre celle-ci indépendante de toute implémentation spécifique.

Deux implémentations concrètes de cette abstraction ont été développées. La première, `OnnxBackend`, permet d'exécuter l'inférence d'un modèle YOLO exporté au format ONNX à l'aide d'ONNX Runtime, aussi bien sur CPU que sur GPU. La seconde, `OakDBackend`, exécute l'inférence directement sur une caméra OAK-D Pro en exploitant son NPU intégré via la bibliothèque DepthAI. Bien que ces deux backends reposent sur des mécanismes d'exécution très différents, ils exposent exactement la même interface côté pipeline, ce qui garantit une utilisation totalement interchangeable au sein du benchmark.

L'un des objectifs majeurs de cette abstraction est la standardisation du pré-traitement et du post-traitement des données. Le pré-traitement inclut le redimensionnement des images ainsi que l'application du letterboxing afin de conserver le ratio d'aspect. Le post-traitement regroupe quant à lui le décodage des sorties du modèle YOLO, l'application des seuils de confiance et du Non-Maximum Suppression (NMS), la génération des bounding boxes et des masques de segmentation, ainsi que la suppression du letterboxing pour reprojeter correctement les résultats sur l'image originale. Cette standardisation garantit que les différences observées lors du benchmark proviennent uniquement du backend d'inférence et du matériel sous-jacent, et non de variations dans le traitement des données.

Grâce à cette approche, la pipeline de benchmark est totalement découpée des implémentations spécifiques d'inférence. Elle peut ainsi être utilisée pour évaluer différents matériels ou frameworks sans nécessiter de modifications structurelles. L'ajout d'un nouveau backend se limite à l'implémentation de la classe abstraite `InferenceBackend`, ce qui améliore la lisibilité, la maintenabilité et l'extensibilité du code, tout en assurant des résultats comparables, reproductibles et méthodologiquement rigoureux.

IV.3. Mesure de la consommation

Pour la caméra OAK-D Pro et le Raspberry Pi, un dongle USB-C est utilisé afin de mesurer la consommation énergétique des appareils. Ce dongle est intercalé entre la source d'alimentation et le dispositif à mesurer. Les données de consommation sont récupérées en filmant l'affichage du dongle pendant toute la durée des benchmarks, puis en traitant la vidéo à l'aide d'un script Python permettant d'extraire un ensemble de valeurs à une fréquence d'une mesure par seconde.

IV.4. Scénarios

Sur la base de ce protocole commun, plusieurs scénarios expérimentaux ont été définis afin d'évaluer le comportement du pipeline d'inférence sur différentes plateformes matérielles.

IV.4.1. Scénario 1 : Caméra OAK-D Pro

Dans ce scénario, la caméra OAK-D Pro est connectée à un ordinateur hôte avec un dongle de mesure de la consommation électrique. L'inférence est exécutée directement sur le VPU embarqué de la caméra avec les images envoyées à la caméra également via le pipeline créé afin de lancer le modèle sur le VPU de la caméra.

IV.4.2. Scénario 2 : Raspberry Pi 4

Ce scénario vise à évaluer l'exécution du modèle sur une plateforme embarquée sans accélération matérielle dédiée.

Le dongle de mesure est placé entre l'alimentation et le Raspberry Pi pour capturer la consommation globale du système. Seul le benchmark est exécuté sur l'appareil, avec un système d'exploitation épuré de toute autre application active. Il est primordial que la consommation hors inférence reste stable et constante tout au long des tests pour garantir la fiabilité des mesures.

IV.4.3. Scénario 3 : Ordinateur portable

Dans ce scénario, un ordinateur portable est utilisé pour exécuter le pipeline d'inférence. La consommation des ressources matérielles est suivie à l'aide d'outils logiciels tels que NVML ou HWiNFO. Ces deux outils se basent sur des résistances intégrées au matériel, un driver peut ensuite récupérer la tension aux bornes de ces résistances pour en déduire l'intensité du courant et donc la puissance globale consommée. Comme pour les autres scénarios, la consommation au repos sera prise en compte afin d'isoler l'impact du traitement du modèle.

V. Les modèles et les optimisations

Dans sa configuration native, le modèle YOLOv11 Nano Segmentation est encodé en FP32 (Floating Point 32-bit) et totalise 2,9 millions de paramètres. Nous avons mis en œuvre plusieurs types d'optimisations pour ensuite comparer leurs effets sur les performances selon nos critères d'évaluation.

V.1. Optimisations générales

V.1.1. Quantification

La quantification est une technique incontournable pour le déploiement sur cibles matérielles contraintes.

Elle consiste en la réduction de la précision numérique des poids et des activations du modèle. En passant d'une représentation à haute précision (généralement 32 bits) à une représentation plus compacte, elle vise trois objectifs majeurs :

- Accélérer l'inférence : Les opérations sur des entiers ou des flottants de taille réduite sont traitées plus rapidement par les unités de calcul.
- Réduire la consommation énergétique : Moins de calculs et de transferts de données impliquent une baisse de la puissance requise.
- Diminuer l'empreinte mémoire : Le modèle occupe moins d'espace de stockage et de RAM, facilitant son intégration sur des dispositifs limités.

Il existe deux approches principales pour appliquer la quantification, chacune présentant des compromis différents :

	Quantification Dynamique	Quantification Statique
Quantification	Hybride Poids quantifiés hors-ligne, mais activations quantifiées à la volée	Hors-ligne Tout est calculé et fixé avant le déploiement
Prérequis	Aucun	Jeu de données de calibration pour fixer les échelles
Vitesse d'inférence	Gains modérés Le calcul des facteurs d'échelle à l'exécution crée une surcharge	Maximale Opérations purement entières, exploitant pleinement les accélérateurs matériels
Empreinte mémoire	Réduite Car nécessite de la mémoire tampon pour les conversions dynamiques	Minimale Le modèle est chargé et exécuté directement en format compressé

Tableau 2. – Comparatif des stratégies de quantification

Dans un souci de simplification du protocole expérimental, nous avons choisi de nous concentrer exclusivement sur la quantification statique dans cette étude.

Nous avons décliné et évalué le modèle YOLOv11 selon trois formats de représentation des données :

- FP32 (Floating Point 32-bit) : Le format standard en simple précision, offrant la plus grande fidélité mais nécessitant plus de ressources.
- FP16 (Floating Point 16-bit) : Une version en demi-précision, réduisant la taille du modèle tout en conservant des poids en virgule flottante.
- INT8 (Integer 8-bit) : Une version compressée utilisant des nombres entiers sur 8 bits, maximisant la vitesse et réduisant l'empreinte mémoire.

La prise en charge de ces formats varie selon le matériel utilisé. Si les architectures classiques (CPU, GPU) et le Raspberry Pi supportent l'ensemble des formats, la caméra OAK-D Pro V2 est limitée exclusivement au format FP16. À noter que cette limitation a été levée sur la version 4 (OAK-D Pro V4), qui supporte davantage de formats.

Plateforme	FP32	FP16	INT8
CPU	✓	✓	✓
GPU	✓	✓	✓
Raspberry Pi	✓	✓	✓
OAK-D Pro V2		✓	

Tableau 3. – Compatibilité des formats par plateforme

V.1.2. Pruning structuré

Contrairement au pruning classique (non-structuré) qui se contente de fixer la valeur de certains poids à zéro sans modifier l'architecture, le pruning structuré supprime physiquement des canaux ou des couches entières du réseau.

Cette méthode présente deux avantages majeurs :

- Le fichier du modèle est allégé de manière effective.
- L'inférence est réellement accélérée, car le matériel de calcul n'a pas besoin de traiter des matrices « creuses » (sparse), mais simplement des matrices plus petites.

Pour mettre en œuvre cette opération complexe, nous nous sommes appuyés sur le projet [YOLO-Pruning-RKNN](#)° de *heyongxin233*. Nous avons généré plusieurs variantes du modèle (voir Tableau 4) en faisant varier le taux de suppression des poids.

Taux de suppression	5%	10%	15%	20%	25%	50%	75%
Paramètres restants (M)	2.755	2.610	2.465	2.320	2.175	1.450	0.725

Tableau 4. – Impact du pruning sur le nombre de paramètres

V.2. Optimisations spécifiques aux plateformes

Afin de tirer parti des architectures spécifiques de chaque plateforme, nous avons adapté les stratégies d'exécution (Providers) et les paramètres d'inférence pour chacune d'entre elles.

Sur la OAK-D Pro, l'optimisation se joue au niveau de l'allocation des ressources du VPU Myriad X. Nous avons configuré l'utilisation de 8 SHAVEs (cœurs vectoriels dédiés au traitement neuronal) sur les 16 disponibles. Selon la documentation officielle, cette répartition constitue le compromis technique idéal : elle maximise la vitesse d'inférence tout en réservant la puissance de calcul nécessaire aux tâches parallèles critiques, telles que la génération de la carte de profondeur et l'encodage du flux vidéo.

Pour l'environnement PC, nous avons distingué deux cas de figure en utilisant des moteurs d'exécution (providers) différents. Pour l'évaluation du processeur seul, nous avons utilisé le *CPUExecutionProvider* standard. En revanche, pour bénéficier de l'accélération matérielle, nous avons sélectionné le *CUDAExecutionProvider*. Ce dernier permet de déporter les calculs sur la carte graphique NVIDIA, tirant ainsi parti du parallélisme massif des cœurs CUDA pour accélérer les opérations matricielles et libérer le processeur central.

Enfin, sur Raspberry Pi 4, bien que l'utilisation du *ACLExecutionProvider* (basé sur l'Arm Compute Library) aurait été préférable pour exploiter les instructions NEON propres à l'architecture ARM, l'indisponibilité d'une version pré-compilée nous a contraints à utiliser le *CPUExecutionProvider*. Pour compenser l'absence d'accélération matérielle dédiée, nous avons activé plusieurs optimisations :

- Graph Optimization : Fusion des couches pour réduire le nombre d'opérations.
- Multi-threading : Exploitation des 4 cœurs physiques pour paralléliser les calculs.
- Gestion Mémoire : Activation de drapeaux spécifiques pour prévenir la saturation de la RAM.
- Mode Séquentiel : Imposition d'un ordre d'exécution strict pour stabiliser la charge CPU.

VI. Benchmark

Nous avons réalisé les benchmarks sur chaque plateforme et obtenu les résultats illustrés dans les figures suivantes.

Pour commencer, la Fig. 1 présente l'évolution de la consommation électrique sur chaque plateforme au cours du benchmark. Afin de permettre une comparaison équitable entre les systèmes, la consommation statique (au repos) a été soustraite de la consommation totale pour ne conserver que la part due à l'inférence des modèles. On peut observer que, pour la caméra et le Raspberry Pi, la consommation est non seulement très faible, mais surtout très similaire d'un modèle à l'autre. Ainsi, ces graphiques permettent principalement de visualiser la variation de la durée des benchmarks selon le modèle, et par extension la variation des temps d'inférence (décrise plus en détail avec la Fig. 3).

On remarque également que les temps d'inférence de la caméra et du Raspberry Pi sont bien supérieurs à ceux du processeur (CPU) et de la carte graphique (GPU) du PC. Enfin, l'examen du graphique du GPU montre que le modèle converti en INT8 requiert beaucoup plus de temps et d'énergie que les autres. À l'inverse, la durée du benchmark pour le modèle élagué à 75 % sur GPU a été nettement plus courte que pour tous les autres modèles.

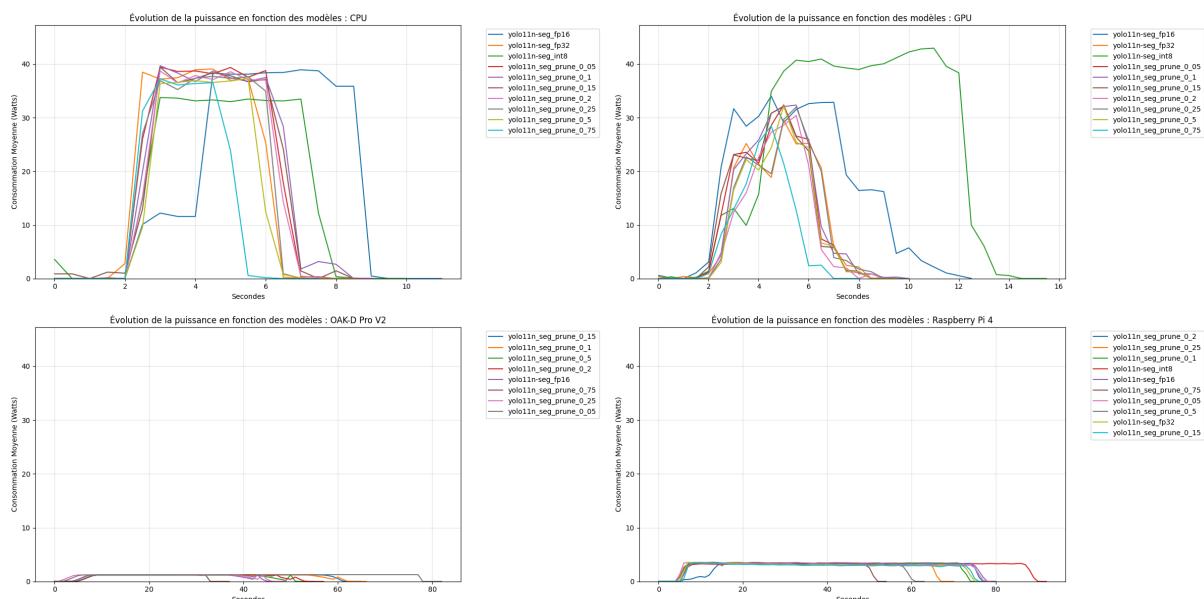


Fig. 1. – Graphiques représentant la consommation au cours du temps en fonction des plateformes

La Fig. 2 présente la consommation moyenne de chaque modèle par plateforme. On y observe des résultats cohérents avec la Fig. 1 : la caméra et le Raspberry Pi consomment moins que la plateforme PC, que ce soit sur CPU ou GPU. On note également qu'en moyenne, le GPU (avec ses potentielles fallback sur CPU) consomme moins que le CPU seul. Cela s'explique par le fait que le GPU est le principal composant s'occupant des calculs et qu'il est optimisé pour les calculs massivement parallèles, ce qui correspond parfaitement à l'architecture du modèle YOLOv11.

Cependant, pour le modèle quantifié en INT8, cette tendance s'inverse : la consommation du CPU pour les fallbacks depuis le GPU devient très élevée tandis que celle du GPU seul est la plus faible de tous les modèles. Ce phénomène s'explique par le fait que la quantification INT8 est réalisée dynamiquement ; des opérations supplémentaires sont alors ajoutées. Ces dernières n'étant pas compatibles avec le GPU, elles entraînent de nombreux renvois vers le CPU (fallback), ce qui augmente significativement la consommation électrique globale.

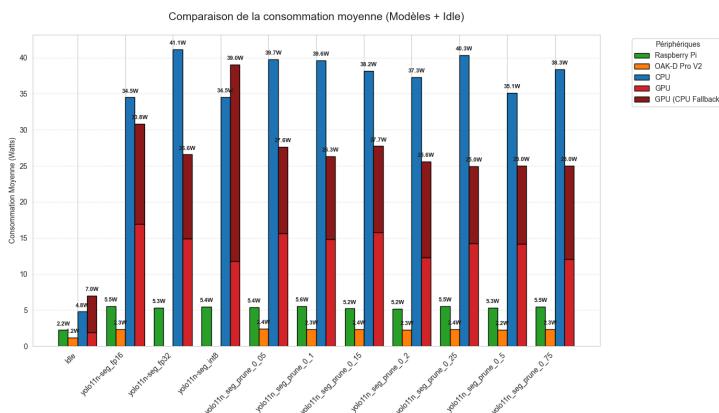


Fig. 2. – Graphique représentant la consommation moyenne en watt de chaque plateforme en fonction des modèles

La Fig. 3 présente le temps d'inférence moyen pour chaque modèle sur chaque plateforme. Comme suggéré par la Fig. 1, le GPU et le CPU sont les plateformes affichant les temps d'inférence les plus courts, tandis que le Raspberry Pi présente les plus longs. Cette différence structurelle s'explique par l'optimisation du GPU pour ce type de calculs. Le processeur du PC, bien que non spécifiquement optimisé, compense par une fréquence d'horloge très élevée, ce qui lui permet d'obtenir des temps d'inférence inférieurs à ceux des deux autres plateformes. En effet, bien que la caméra dispose d'un VPU (Vision Processing Unit) propice aux calculs parallèles, sa fréquence de fonctionnement reste relativement basse. Enfin, le Raspberry Pi, équipé d'un simple processeur ARM, n'est pas optimisé pour ces tâches intensives.

Une fois de plus, on remarque que pour le modèle quantifié en INT8, le GPU affiche un temps d'inférence plus élevé que le CPU, ce qui peut paraître contre-intuitif. Comme mentionné précédemment, cela est dû au fait que certaines opérations ne sont pas prises en charge par le GPU et sont renvoyées au CPU. Cela impose des transferts de données récurrents entre le CPU et le GPU durant l'inférence, ralentissant ainsi l'ensemble du processus.

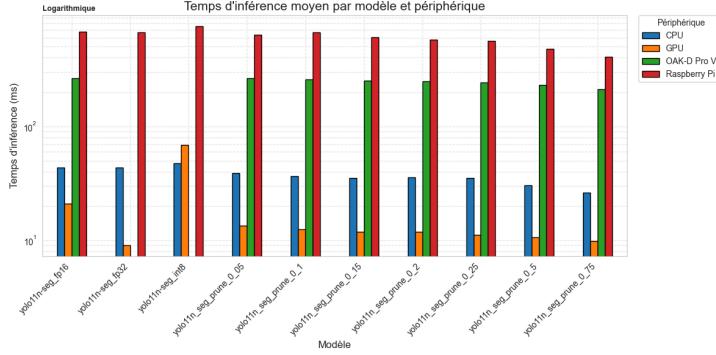


Fig. 3. – Graphique représentant le temps d'inférence moyen en millisecondes de chaque plateforme en fonction des modèles

Enfin, la Fig. 4 résume l'ensemble des métriques capturées en mettant en relation la mAP (Mean Average Precision) avec l'énergie consommée. L'énergie est ici calculée en multipliant le temps d'inférence moyen (converti en secondes) par la consommation moyenne, permettant d'obtenir un résultat en Joules. Deux graphiques sont présentés : l'un corrèle la mAP des boîtes de détection et l'autre la mAP des masques de segmentation.

Sur ces deux graphiques, on observe deux groupes distincts concernant le score mAP : d'un côté les modèles élagués (pruned) et de l'autre les modèles quantifiés. Cet écart vient du fait que les modèles élagués nécessitent un ré-entraînement. Nous avons limité celui-ci à une quinzaine de minutes mais un temps d'entraînement plus long aurait sans doute permis d'obtenir de meilleurs résultats.

Si l'on exclut les modèles quantifiés en INT8 et le modèle en FP32, l'analyse de la consommation énergétique permet de distinguer deux groupes. Le premier, constitué de la caméra et du GPU, se montre le plus efficace : la caméra bénéficie d'un temps d'inférence très court, tandis que le GPU offre un bon compromis entre consommation et rapidité. Le second groupe, réunissant le Raspberry Pi et le CPU, affiche une consommation par inférence nettement plus élevée. Cela s'explique par un calcul séquentiel qui s'avère soit très coûteux en puissance (pour le CPU), soit excessif en temps d'exécution (pour le Raspberry Pi).

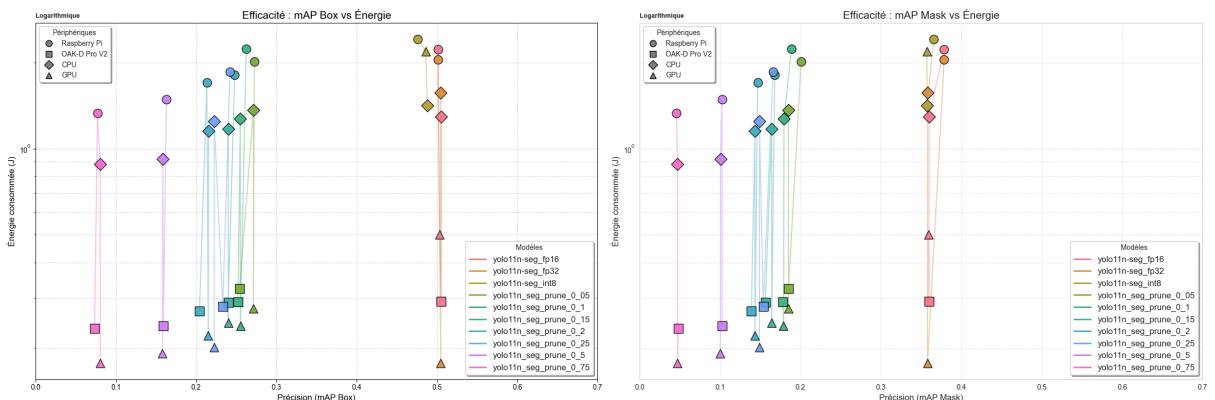


Fig. 4. – Graphiques représentant la consommation énergétique en joule de chaque plateforme en fonction du score mAP évalué sur les boîtes de détection et les masques de segmentation

VII. Conclusion

Grâce à l'ensemble des métriques obtenues durant les benchmarks et à l'étude des résultats réalisée précédemment, nous pouvons déterminer la solution optimale en combinant la plateforme et le modèle les plus adaptés.

En effet, pour notre cas d'usage initial (un robot suiveur de personne) le meilleur choix de plateforme est sans conteste la caméra Luxonis Oak-D Pro V2. Elle offre des résultats de précision (mAP) similaires aux autres plateformes pour tous les modèles testés, qu'il s'agisse des boîtes de détection ou des masques de segmentation.

De plus, son temps d'inférence moyen, compris entre 229 ms et 263 ms selon les modèles, est tout à fait acceptable. Cette latence permet de mettre à jour la position de la cible environ 3 fois par seconde, ce qui est suffisant pour assurer un suivi fluide en temps réel.

Enfin, pour un robot fonctionnant sur une batterie à capacité limitée, la sobriété énergétique est un enjeu crucial. À cet égard, la caméra s'impose comme la plate-forme la plus efficiente, affichant la consommation électrique la plus faible, que ce soit durant les phases d'inférence ou au repos.

	Consommation moyenne	Temps d'inférence	Efficacité énergétique
PC (GPU)	Élevée	Très Rapide	Excellente
OAK-D Pro	Très Faible	Moyen / Lent	Excellente
PC (CPU)	Très élevée	Rapide	Faible
Raspberry Pi 4	Très Faible	Très Lent	Très Faible

Tableau 5. – Synthèse comparative des plateformes

L'ensemble de ces facteurs confirme que la caméra constitue le choix de plateforme le plus rationnel. Toutefois, cette décision doit être couplée à la sélection d'un modèle dont l'optimisation est en parfaite adéquation avec les capacités du matériel.

À cet égard, le modèle de base quantifié en FP16 (format requis pour l'exécution sur le VPU de la caméra) apparaît comme la solution la plus performante, car il offre les meilleurs scores mAP. Par ailleurs, bien qu'un modèle élagué (pruned) permettrait de réduire le temps d'inférence de quelques dizaines de millisecondes, cette option nécessiterait un ré-entraînement prolongé pour garantir une précision équivalente au modèle FP16. Étant donné que la latence actuelle du modèle FP16 n'est pas rédhibitoire pour notre application, la priorité est accordée au maintien de la précision maximale plutôt qu'à un gain de temps marginal.

	Précision	Temps d'inférence
Modèle fortement élagué (Pruned 75%)	Très faible	Très rapide
Modèle faiblement élagué (Pruned 5%)	Faible	Rapide
Modèle fortement quantifié (INT8)	Élevée	Moyen / Lent (fallback)
Modèle faiblement quantifié (INT16)	Très élevée	Moyen
Modèle standard (FP32)	Très élevée	Moyen

Tableau 6. – Synthèse comparative des modèles

En conclusion, le binôme formé par la caméra OAK-D Pro V2 et le modèle YOLOv11n quantifié en FP16 constitue le choix optimal pour notre projet. Cette décision est validée par l'ensemble des benchmarks réalisés et l'analyse rigoureuse des métriques capturées lors de cette étude.