



Rapport de Stage

—

FR4IAV Vision Project

BARSOT Thomas



INSA Rouen – 2024-2025

Table des matières

1	Entreprise	6
1.1	L'entreprise	7
1.2	Le lieu de stage	8
2	Sujet	9
2.1	Contexte du stage	10
2.2	Objectifs du stage	11
2.3	Contraintes et périmètre	11
3	Travail effectué	12
3.1	Analyse du besoin	13
3.1.1	Lecture et compréhension du cahier des charges	13
3.1.2	Identification des enjeux techniques	14
3.2	Propositions et choix de solution	15
3.2.1	Comparaison de plusieurs solutions : annotations	15
Annotations manuelles	15	
Annotations automatiques	15	
3.2.2	Comparaison de plusieurs solutions : type de modèle	15
Modèles de détection classiques	15	
Modèles orientés	16	
Modèles en temps réel	16	
3.2.3	Comparaison de plusieurs solutions : type d'entraînement	17
Modèles réutilisés	17	
Modèles entraînés from scratch	17	
Modèles finetunés	17	
3.2.4	Avantages / Inconvénients	18
3.2.5	Choix de la solution retenue	18
3.3	Description détaillée de la solution retenue	19
3.3.1	Architecture générale	19
Modèle de détection classique	19	
Modèle personnalisé : Angle, vitesse et distance	20	
Filtre de Kalman	20	
3.3.2	Modules développés	20
Module d'annotation automatique	21	
Module d'entraînement	21	
Module RTMaps	21	
3.3.3	Technologies et outils utilisés	22
3.4	Mise en œuvre technique	22

3.4.1	Développement	22
3.4.2	Tests et validation	23
3.4.3	Difficultés rencontrées	23
3.5	Résultats obtenus	24
3.5.1	Présentation des résultats	24
Annotations automatiques	24	
Annotations corrigées	25	
Benchmarks de détection	26	
Benchmarks de prédiction d'angle, de vitesse et de distance	28	
Module d'inférence complet	29	
Module RTMaps	29	
3.5.2	Analyse critique	30
3.5.3	Axes d'amélioration	31
4	Dév. Durable	32
4.1	Impacts environnementaux	33
4.2	Aspects sociaux et éthiques	33
4.3	Bonnes pratiques mises en œuvre	33
5	Conclusion	34
5.1	Bilan du stage	35
5.2	Évolutions possibles	35
6	Bibliographie	36
6.1	Bibliographie	37
7	Annexes	38
7.1	Benchmarks	39
7.1.1	Benchmarks complets des modèles YOLO	39
7.1.2	Benchmarks complets des modèles RTMDet	48
7.1.3	Benchmarks de modèle angle, vitesse et distance	52
7.2	RTMaps	56
7.3	Interface Streamlit	61
7.3.1	Application d'annotation	61
7.3.2	Application d'inférence	63
8	Fin	65
8.1	Résumé	66
8.2	Abstract	66

Remerciements

Avant de conclure ce travail, je souhaite exprimer ma gratitude envers celles et ceux qui m'ont accompagné et soutenu tout au long de ce projet.

Je remercie tout d'abord Rodolphe DEBEAUVAIS, pour sa disponibilité, sa confiance, et l'encadrement rigoureux qu'il m'a offert durant cette expérience. Ses conseils ont été précieux tout au long du projet.

Je tiens également à remercier Massimiliano BALESTRERI, PDG de l'entreprise, pour m'avoir permis de m'impliquer pleinement dans un projet ambitieux, et pour l'intérêt qu'il a porté à mes contributions.

Je remercie Tarik TAQUI, doctorant, pour nos échanges riches et pertinents, ainsi que pour sa bienveillance et son accompagnement dans la compréhension des aspects les plus pointus de la vision embarquée.

Je remercie chaleureusement le laboratoire LITIS ainsi que ses doctorants et stagiaires pour son accueil dans un cadre de travail stimulant, et le professeur Abdelaziz BENS-RHAIR pour m'avoir orienté vers ce stage avec bienveillance et implication.

Je souhaite également remercier l'INSA, et tout particulièrement Benoît GAÜZERE, responsable des stages, pour son accompagnement et sa réactivité tout au long du processus.

À tous, merci pour votre confiance, votre soutien, et pour m'avoir offert les conditions idéales pour mener à bien ce projet. Cette expérience a été formatrice, autant sur le plan technique que sur le plan humain.

Lexique

Terme	Définition
GT (Ground Truth)	Données de référence considérées comme la vérité terrain. Utilisées pour évaluer la qualité ou la précision des prédictions d'un modèle ou d'un algorithme.
ÉPOQUE (Epoch)	En apprentissage automatique, une époque correspond à un passage complet de l'ensemble des données d'entraînement à travers le modèle. Plusieurs époques sont nécessaires pour l'optimisation.
RTMAPS	Logiciel temps réel permettant de collecter, traiter et synchroniser des données multi-capteurs (caméras, LiDAR, radars, etc.), utilisé notamment dans le domaine automobile et robotique.
YAW (ORIENTATION)	Orientation du véhicule, dans notre cas d'usage, elle est exprimée par l'angle entre son axe longitudinal et le plan du circuit. Permet de décrire la direction vers laquelle se dirige la voiture.

Chapitre 1

Présentation de l'entreprise et du lieu de stage

Sommaire

1.1	L'entreprise	7
1.2	Le lieu de stage	8

1.1 L'entreprise

Le stage a été réalisé pour le compte de l'entreprise **ALADIN**, un consortium technologique français innovant rassemblant huit PME spécialisées dans les hautes technologies. Sa mission principale est de favoriser l'intégration des innovations électroniques et numériques dans les véhicules, afin de répondre aux enjeux de sécurité routière et d'amélioration du confort de conduite.

ALADIN s'inscrit dans le domaine en pleine croissance des systèmes avancés d'aide à la conduite (ADAS) et des véhicules autonomes. L'entreprise accompagne ses clients tout au long de leur démarche de création et d'intégration de nouvelles solutions ADAS, en s'appuyant sur :

- son expertise multidisciplinaire, acquise dans des domaines complémentaires (électronique, traitement de données, intelligence artificielle, robotique),
- des technologies ouvertes, modulaires et interopérables,
- une capacité à évaluer et intégrer, lorsque nécessaire, les technologies déjà disponibles sur le marché.

Leur approche repose sur la création de solutions clés en main à forte valeur ajoutée pour ses clients, notamment dans les secteurs de l'automobile, de la mobilité intelligente et de la recherche appliquée.

Expertise technologique

En résumé, cette entreprise se distingue par une expertise poussée dans les domaines suivants :

- **Technologie** : perception, localisation, planification, contrôle, automatisation, connectivité, simulation, intégration.
- **Développement industriel** : gestion de projets de TRL 3 à 9, conception industrielle, modélisation, fusion de données, algorithmes de deep learning.
- **Stratégie** : intégration ouverte de technologies, veille technologique, exploitation d'un écosystème riche à l'échelle française et internationale.
- **Méthodologie** : culture de la collaboration, gestion de projets agile, accompagnement de pilotes, montée en compétences des équipes partenaires.



FIGURE 1.1 – Logo ALADIN

1.2 Le lieu de stage

Le stage s'est déroulé au sein du laboratoire LITIS, à Saint-Etienne du Rouvray. De nombreux chercheurs et doctorants y travaillent, permettant de nombreuses discussions et une entraide collective. Un autre stagiaire actuellement sur le projet était présent, GNAOUI Ziyad ainsi qu'un doctorant, TAOUI Tarik, favorisant une étroite collaboration sur le projet. Ainsi, j'ai pu réaliser ce stage dans ces locaux, à raison de 5 jours par semaine, hors fermeture durant août.

De manière plus globale, ce stage a été encadré par DEBEAUVAIS Rodolphe, Project Manager chez Aladin. Il a assuré le suivi du stage hebdomadairement afin de discuter des solutions proposées et de l'avancement.



FIGURE 1.2 – Logo LITIS

Chapitre 2

Présentation du sujet du stage

Sommaire

2.1	Contexte du stage	10
2.2	Objectifs du stage	11
2.3	Contraintes et périmètre	11

2.1 Contexte du stage

Le stage s'inscrit dans le cadre du projet A2RL, dont l'objectif est le développement de systèmes de perception pour véhicules autonomes de compétition. Ce projet vise à permettre à un véhicule autonome de concourir sur circuit contre d'autres nations.

Dans ce contexte, le rôle du système ALADIN est central : il constitue la plateforme logicielle et matérielle de perception embarquée qui collecte, traite et interprète les données sensorielles (les images dans le cadre de ce stage) pour détecter les monoplaces adverses en temps réel. ALADIN facilite ainsi la prise de décision autonome en fournissant une estimation précise et rapide de l'environnement proche du véhicule.



FIGURE 2.1 – Véhicules autonomes A2RL

La problématique traitée lors de ce stage concerne spécifiquement la détection automatique de monoplaces de course à partir d'images simulées, en exploitant les données de vérité terrain fournies par le simulateur Autonoma.

Ce travail intervient en amont de l'entraînement des modèles d'intelligence artificielle intégrés dans ALADIN, en constituant un dataset robuste, réaliste et parfaitement annoté pour la tâche de détection d'objets. Ce dataset améliore la fiabilité et la performance des algorithmes embarqués dans ALADIN, contribuant ainsi directement aux capacités autonomes des véhicules du projet A2RL.

2.2 Objectifs du stage

Les objectifs du stage s'inscrivaient dans une démarche globale visant à renforcer les capacités de perception du véhicule autonome. De manière générale, il s'agissait de :

- Développer des méthodes permettant d'améliorer la perception et la compréhension de l'environnement par la voiture.
- Concevoir des outils et des modèles pouvant être intégrés et utilisés en temps réel à bord du véhicule.
- Préparer et structurer les données nécessaires pour entraîner et évaluer des modèles de détection fiables.
- Évaluer et comparer différentes approches de perception afin de retenir les plus adaptées au contexte applicatif.
- Valider la démarche par des tests en conditions proches du réel, avec une mise en situation prévue lors d'essais grandeur nature sur circuit à Abu Dhabi.

2.3 Contraintes et périmètre

Le stage s'est déroulé sur une période limitée, ce qui imposait un cadrage strict des livrables à produire. Certaines contraintes ont été identifiées dès le début :

- **Contraintes temporelles** : Durée limitée du stage, nécessitant une organisation rigoureuse.
- **Contraintes matérielles** : Pas de GPU au début du stage, retardant les phases d'entraînements.
- **Contraintes fonctionnelles** : Le stage ne portait que sur la détection (pas de segmentation).
- **Contraintes de format** : Les annotations devaient respecter des standards (COCO, KITTI) pour être exploitables par des modèles IA. Les résultats doivent être visionnables facilement et intuitivement.
- **Contraintes logicielles** : Utilisation d'un logiciel complexe : RTMaps, nécessitant une phase d'exploration et de prise en main.

Chapitre 3

Travail effectué

Sommaire

3.1 Analyse du besoin	13
3.1.1 Lecture et compréhension du cahier des charges	13
3.1.2 Identification des enjeux techniques	14
3.2 Propositions et choix de solution	15
3.2.1 Comparaison de plusieurs solutions : annotations	15
3.2.2 Comparaison de plusieurs solutions : type de modèle	15
3.2.3 Comparaison de plusieurs solutions : type d'entraînement	17
3.2.4 Avantages / Inconvénients	18
3.2.5 Choix de la solution retenue	18
3.3 Description détaillée de la solution retenue	19
3.3.1 Architecture générale	19
3.3.2 Modules développés	20
3.3.3 Technologies et outils utilisés	22
3.4 Mise en œuvre technique	22
3.4.1 Développement	22
3.4.2 Tests et validation	23
3.4.3 Difficultés rencontrées	23
3.5 Résultats obtenus	24
3.5.1 Présentation des résultats	24
3.5.2 Analyse critique	30
3.5.3 Axes d'amélioration	31

3.1 Analyse du besoin

3.1.1 Lecture et compréhension du cahier des charges

Le cahier des charges précise la nécessité de développer un système fiable de détection automatique de voitures autonomes en contexte de compétition, à partir d'images majoritairement simulées et par la suite réelles. Il met l'accent sur la précision des annotations, la robustesse des modèles et la compatibilité avec les systèmes embarqués, notamment avec l'utilisation de RTMaps.

Voici alors le cahier des charges initial :

- Analyser les capacités d'export de ground truth du simulateur Autonoma.
- Développer un script pour associer chaque image caméra à sa bounding box de monoplace issue de la vérité terrain.
- Générer automatiquement les fichiers d'annotations au format standard (ex : COCO, KITTI).
- Construire un dataset d'images annotées pour un usage machine learning.
- Vérifier la qualité et la cohérence des données produites.
- Proposer un système simple de visualisation/contrôle des résultats en temps réel
- Documenter les outils et méthodes développés.

Additionnellement, il a été établi lors du stage qu'il serait utile de pouvoir détecter la trajectoire, l'angle, la vitesse et la distance de la voiture, si le temps le permettait.

Dans l'ordre de réalisation nous pourrions le résumer en un schéma :

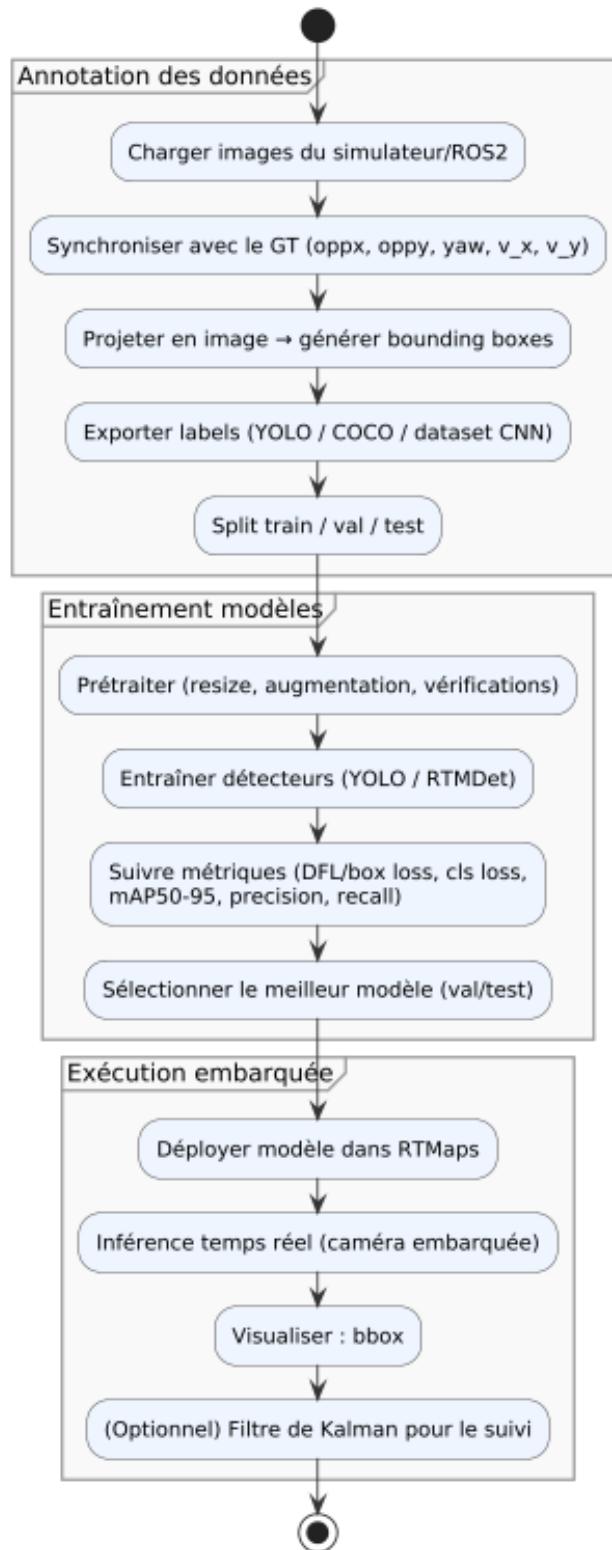


FIGURE 3.1 – Cahier des charges

3.1.2 Identification des enjeux techniques

Parmi les enjeux majeurs, on note la gestion de données hétérogènes, la nécessité d'annotations précises et automatisées, la limitation des ressources embarquées, et la nécessité d'un traitement temps réel pour garantir la réactivité du véhicule autonome.

3.2 Propositions et choix de solution

3.2.1 Comparaison de plusieurs solutions : annotations

Annotations manuelles

Les annotations manuelles garantissent une qualité élevée mais sont coûteuses en temps et peu scalables. Cette méthode n'est donc pas privilégiée sur les données simulées, mais reste envisageable afin de les améliorer. De plus, elles seront nécessaires sur de potentielles données réelles, qui ne pourraient pas être générées automatiquement.

Annotations automatiques

Les annotations automatiques permettent de générer rapidement un grand volume de données annotées, notamment grâce au simulateur Autonoma, mais requièrent un contrôle qualité rigoureux et peuvent être moins précises.

De plus, elles permettent d'être génériques, maintenables et modifiables facilement.

Cette méthode paraît donc optimale dans un cas où le simulateur peut nous fournir les informations nécessaires à la labellisation de chaque image.

3.2.2 Comparaison de plusieurs solutions : type de modèle

Concernant l'entraînement, il existe différents types de modèles de détection, nous devons donc choisir le plus adapté à notre cas d'usage.

Modèles de détection classiques

Un modèle de détection d'objets est un réseau de neurones profond conçu pour localiser et classifier plusieurs objets dans une image. Ces modèles sont composés de plusieurs parties clés, permettant d'extraire des caractéristiques pertinentes et de prédire les boîtes englobantes ainsi que les classes associées. Il est composé de trois blocs principaux :

- **Backbone** : C'est un réseau convolutionnel pré-entraîné (comme ResNet, VGG ou EfficientNet) qui sert à extraire des représentations visuelles riches à partir de l'image d'entrée. Il transforme l'image brute en cartes de caractéristiques (feature maps) de plus en plus abstraites et robustes aux variations visuelles.
- **Neck** : Cette partie intermédiaire fusionne et affine les caractéristiques extraites par le backbone à différentes échelles. Elle améliore la capacité du modèle à détecter des objets de tailles variées. Des modules courants pour le neck incluent le Feature Pyramid Network (FPN) ou le Path Aggregation Network (PAN), qui combinent des informations hiérarchiques issues du backbone.
- **Head** : Le head est responsable de la prédiction finale. Il génère les bounding boxes (boîtes englobantes) qui localisent les objets, ainsi que les scores de classification associés. Selon l'approche, il peut aussi prédire des paramètres additionnels comme la segmentation ou la profondeur.

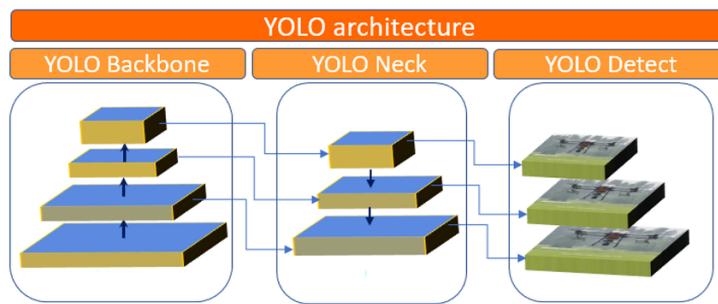


FIGURE 3.2 – Basic architecture of the YOLO series network

Ces modèles permettent donc la détection d'objets de manière générique et efficace, en exploitant une architecture modulaire adaptée à de nombreux contextes. Dans le cadre de ce projet, ils sont particulièrement utiles pour la détection d'objets spécifiques à l'environnement de la voiture autonome comme ses adversaires, et fournir au véhicule une perception fiable et exploitable en temps réel.

Modèles orientés

Les modèles orientés, sont similaires aux modèles de détection classique, mais intègrent la notion d'objet orientés, mais offrant uniquement une rotation dans le plan de la caméra, ce qui est incompatible avec le besoin de détecter une trajectoire ou un angle sur le plan du circuit.

Modèles en temps réel

Les modèles temps réel intègrent la dimension temporelle, ce qui aurait pu être utile pour prédire des trajectoires, cependant ces modèles sont trop coûteux pour réaliser du temps réel, cet axe a donc été écarté aussi, il est plus optimisé de travailler avec un filtre de Kalman, que nous abordons : ici.

3.2.3 Comparaison de plusieurs solutions : type d'entraînement

Afin d'entraîner correctement le modèle que nous avons choisi, trois possibilités s'offrent à nous :

Modèles réutilisés

Nous pourrions réutiliser des poids déjà entraînés sur des datasets urbains, comme KITTI, qui permet de détecter de nombreux véhicules, cependant ce dataset n'est pas du tout entraîné sur des données de voitures sportives, comme la SuperFormula que nous utilisons au sein du projet. Cette méthode a donc été écartée.

Modèles entraînés from scratch

L'entraînement from scratch consiste à initialiser un réseau de neurones sans connaissances préalables et à l'entraîner intégralement sur un jeu de données spécifique. Cette approche offre une liberté totale dans la conception du modèle et permet d'obtenir des représentations entièrement adaptées au domaine ciblé. Cependant, elle nécessite un volume considérable de données annotées ainsi qu'un coût computationnel important (temps d'entraînement et ressources matérielles). De plus, sans données massives et variées, le risque de surapprentissage est élevé et la capacité de généralisation peut être limitée. Dans notre cas, il est pertinent de la soumettre à un benchmark afin d'en évaluer les performances.

Modèles finetunés

Le fine-tuning s'appuie sur des modèles pré-entraînés sur de vastes bases de données généralistes (par exemple ImageNet ou COCO). Ces modèles ont déjà acquis une capacité à extraire des caractéristiques visuelles pertinentes et robustes, qui peuvent ensuite être adaptées à une nouvelle tâche ou à un domaine spécifique. Cette approche réduit considérablement le coût d'entraînement et améliore la convergence, tout en profitant d'une généralisation issue du pré-entraînement.

Dans notre cas, il est particulièrement adapté : il permet de spécialiser un modèle déjà performant aux contraintes propres à la voiture de course (environnement de circuit, objets d'intérêt spécifiques, conditions visuelles particulières), tout en conservant la majorité de l'extraction de caractéristiques déjà apprise. Ainsi, le modèle bénéficie à la fois de la robustesse du pré-entraînement et de l'adaptation fine aux besoins du projet.

3.2.4 Avantages / Inconvénients

TABLE 3.1 – Comparatif des méthodes d’annotation

Solution	Avantages	Inconvénients
Annotations manuelles	Haute qualité	Très chronophage, peu scalable
Annotations automatiques	Rapide, scalable	Qualité variable, nécessite validation

TABLE 3.2 – Types de modèles de détection

Type de modèle	Avantages	Inconvénients
Modèles classiques	Simplicité et Efficacité	Pas de prédiction d’angle/vitesse/distance/trajectoire
Modèles orientés	Spécificité	Pas adapté à notre cas d’usage
Modèles temporels	Rapide, adapté trajectoire	Modèles plus lourd, pas adapté au temps réel

TABLE 3.3 – Types d’entraînement

Type d’entraînement	Avantages	Inconvénients
Modèle réutilisé	Très rapide à mettre en oeuvre, pas d’entraînement à faire	Aucune personnalisation sur nos données, pas précis
From scratch	Personnalisation	Très coûteux en temps et données
Fine-tuning	Adaptation rapide	Dépend des modèles de base

3.2.5 Choix de la solution retenue

La solution retenue combine l’annotation automatique basée sur le simulateur Autonomia et la réalisation d’un module d’inférence, utilisant des modèles fine-tunés de détection classiques pour assurer un bon compromis entre rapidité, précision et scalabilité.

3.3 Description détaillée de la solution retenue

3.3.1 Architecture générale

Le système comprend une chaîne complète allant de la génération et annotation automatique des données, au pré-traitement, entraînement des modèles de détection et inférence temps réel dans l'environnement embarqué.

L'architecture générale du module d'inférence peut être présentée de la manière suivante :

- Modèle de détection classique (YOLO, RTMDet)
- Modèle personnalisé pour prédire l'angle , la vitesse et la distance
- Filtre de Kalman afin de suivre les objets entre les images (Optionnel)

Le tout devra, idéalement, être intégré dans le logiciel RTMaps afin de pouvoir procéder en temps réel, parallèlement à l'utilisation des autres capteurs au sein de la voiture.

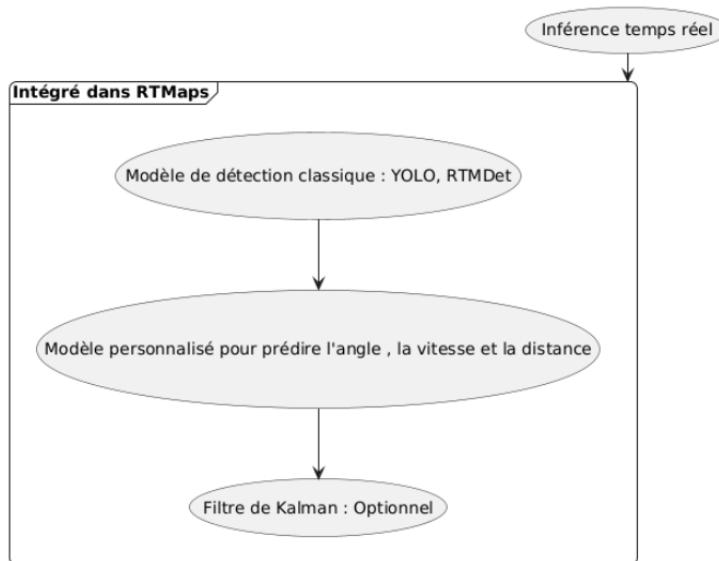


FIGURE 3.3 – Architecture du module d'inférence

Modèle de détection classique

Le modèle de détection classique va nous permettre de prendre en entrée du réseau de neurones une image, et d'obtenir en inférant dessus sa bounding box. Ainsi cette partie de la pipeline se concentre uniquement sur la détection des voitures adverses. Nous allons donc réaliser un benchmark de différents modèles de la famille YOLO et RTMDet (modèle de MMDetection), afin de déterminer quelle architecture et quelle méthode d'entraînement donnent le résultat le plus optimal possible.

Ces deux familles ont été identifiées à l'aide de premiers benchmarks, issus de plusieurs papiers de recherche, disponibles : ici. Ces papiers nous ont permis de réaliser un premier grand tri, afin de se focaliser sur les familles de modèles de détection les plus pertinentes.

Les résultats de notre benchmark sont disponibles : ici.

Modèle personnalisé : Angle, vitesse et distance

Comme nous l'avons vu précédemment, le modèle de détection classique ne va pas prédire l'angle, la vitesse ou encore la distance. Nous avons donc fait le choix de lui associer un second réseau de neurones, plus petit, qui sera chargé d'utiliser les prédictions du modèle de détection pour prédire à son tour l'angle et la vitesse.

Cette partie n'a pas été la priorité, elle a été réalisée en plus afin d'apporter plus d'informations à notre véhicule à partir de la caméra. Cependant, dans les faits, la caméra n'étant pas le seul capteur, une fusion de capteurs serait plus performante pour réaliser cette tâche.

De manière similaire, afin de privilégier les performances et chercher à les maximiser, nous avons réalisé un benchmark de différents backbone afin de choisir le meilleur.

Les résultats de notre benchmark sont disponibles : ici.

Filtre de Kalman

Le filtre de Kalman est un algorithme de filtrage récursif optimal pour l'estimation d'états variables dans le temps à partir d'observations bruitées. Appliqué au suivi d'objets en vision, il permet de maintenir une estimation robuste de la position, de la vitesse et d'autres variables d'état des véhicules détectés sur plusieurs images successives.

Concrètement, le filtre de Kalman réalise deux étapes principales :

- **Prédiction** : projection de l'état courant (position, vitesse, orientation, distance) à l'instant suivant, en utilisant un modèle dynamique.
- **Mise à jour** : correction de cette prédiction à partir des nouvelles observations issues du modèle de détection, afin de réduire l'incertitude.

Grâce à ce mécanisme, le filtre de Kalman facilite l'association des détections successives à des objets spécifiques, permettant ainsi de suivre chaque véhicule de façon continue dans le temps et d'estimer sa trajectoire future.

Il est important de souligner que l'application du filtre de Kalman ne se limite pas aux seules données issues de la caméra. Il peut intégrer d'autres sources d'information, comme les mesures des capteurs embarqués (GPS, IMU, lidar, etc.), afin de construire une estimation plus robuste et multimodale de l'état du véhicule et de son environnement.

Cette description illustre donc l'architecture générale et globale d'un système de suivi multi-capteurs. Dans le cadre de ce stage cependant, l'objectif n'était pas de développer l'ensemble de cette architecture, mais de se concentrer sur la partie perception par vision et sur l'intégration en temps réel dans le flux caméra uniquement.

3.3.2 Modules développés

Bien que le module d'inférence est le module central du projet, qui détecte les opposants et donne de précieuses informations, son bon fonctionnement dépend d'autres modules :

- Module d'annotation automatique exploitant les données de vérité terrain du simulateur Autonoma.
- Module d'entraînement basé sur Ultralytics YOLO et MMDetection.
- Module RTMaps pour l'intégration et la gestion du traitement temps réel sur la plateforme embarquée.

Module d'annotation automatique

La constitution d'un jeu de données annoté de qualité est une étape cruciale pour l'entraînement de modèles performants. Plutôt que de recourir à des annotations manuelles coûteuses et peu scalables, nous avons exploité le ground truth fourni par le simulateur Autonoma. Ce choix a été guidé par la nécessité d'obtenir des annotations précises et cohérentes, notamment les coordonnées relatives opp_x , opp_y , les vitesses vx , vy , ainsi que l'orientation yaw et la *distance* entre la caméra embarquée et le véhicule détecté.

De plus, il a été nécessaire de procéder à une calibration de la caméra afin de projeter correctement les informations issues du simulateur sur l'image, et ainsi générer des annotations cohérentes et exploitables.

Toutefois, cette approche a été limitée par la disponibilité d'un seul véhicule détecté dans les données, dans des situations très similaires tout au long de l'enregistrement, ainsi que par la restriction à une seule caméra frontale. Cela nous a conduit à identifier une piste d'amélioration majeure : étendre la capacité du système à gérer des scénarios multi-véhicules et à exploiter également les caméras latérales, afin de mieux refléter la complexité du milieu réel.

Module d'entraînement

La mise en place du processus d'entraînement a nécessité une série de décisions techniques stratégiques, notamment le choix des architectures adaptées (comme YOLO ou RTMDet), la définition d'hyperparamètres pertinents, et la conception de scripts automatisés pour garantir la reproductibilité et l'efficacité des expériences.

Cette étape a aussi impliqué une validation rigoureuse des performances sur un jeu de validation, afin de s'assurer que le modèle converge correctement et généralise bien aux données réelles. L'objectif était d'obtenir un compromis optimal entre précision, robustesse et temps de calcul, indispensable pour une application embarquée.

Module RTMaps

L'intégration finale dans RTMaps représente un pivot essentiel pour passer de la phase de recherche à l'application embarquée opérationnelle. La démarche a consisté à adapter et encapsuler le modèle entraîné dans un module compatible avec l'architecture temps réel du véhicule, en assurant une communication fluide avec les autres composants logiciels.

Nous avons tiré parti d'une formation spécialisée sur RTMaps pour maîtriser ses fonctionnalités avancées, ce qui a permis d'optimiser l'implémentation et de réduire les temps de latence, garantissant ainsi un fonctionnement fiable dans le contexte exigeant de la course automobile.

Cette approche méthodique, divisée en plusieurs modules, oblige rigueur et générnicité, afin de pouvoir maintenir ce code dans le temps, et l'adapter facilement. Il illustre la combinaison d'une exploitation efficace des ressources disponibles, et d'une intégration logicielle adaptée à l'embarqué, afin de construire une solution robuste et performante, plus concrète.

3.3.3 Technologies et outils utilisés

Dans ce projet, plusieurs technologies et outils clés ont été mobilisés afin de répondre aux exigences de détection et de traitement embarqué :

- **Python** : Langage principal pour le développement des algorithmes et scripts
- **Ultralytics YOLO** : Framework de détection d'objets, réputé pour sa rapidité et précision, utilisé pour l'entraînement et l'inférence des modèles de détection.
- **MMDetection** : Boîte à outils open source puissante pour la détection d'objets, offrant une grande modularité dans la configuration des modèles et pipelines.
- **RTMaps** : Plateforme dédiée à la gestion et au traitement temps réel des données embarquées, facilitant la synchronisation des capteurs et la mise en œuvre des algorithmes sur véhicule.
- **ROS2** : Framework open-source de robotique qui fournit une architecture distribuée pour la communication, le contrôle et l'orchestration de robots via des nœuds interconnectés et un middleware temps réel.

3.4 Mise en œuvre technique

3.4.1 Développement

Le développement a été mené de manière itérative avec un suivi hebdomadaire, permettant d'ajuster rapidement les fonctionnalités en fonction des résultats intermédiaires et des contraintes rencontrées. Cette approche incrémentale a favorisé une montée en puissance progressive des modules et une meilleure maîtrise du périmètre technique.

La première étape a consisté en l'annotation des données. Cela a nécessité une prise en main complète de ROS², ainsi que du format `rosbag/mcap`, afin de rejouer des enregistrements de flux caméra et de synchroniser ces images avec les données de GT fournies par le système Autonoma. La complexité provenait du fait que le GT ne concernait qu'un seul véhicule par image, et qu'une synchronisation stricte entre le timestamp de l'image et les données GT était impérative pour garantir la qualité des annotations.

Une fois ce processus d'annotation automatisé mis en place, l'étape suivante a été la réalisation des benchmarks sur différents modèles de détection et de régression (*YOLOv8* à *YOLOv12* et *RTMDet*), afin de déterminer le compromis optimal entre précision, vitesse d'inférence et taille du modèle.

Pour cela, un jeu de métriques spécifiques a été défini, incluant :

- **Loss DFL (Distribution Focal Loss)** : mesure la qualité de la régression des coordonnées des boîtes englobantes, en améliorant la précision des prédictions.
- **Loss de classification** : évalue la capacité du modèle à prédire correctement la classe de l'objet détecté (ici, le véhicule).
- **mAP50-95** : moyenne des précisions moyennes pour des seuils IoU allant de 0.5 à 0.95, indicateur global de performance en détection.
- **Précision et rappel** sur la détection de véhicules : permettent d'évaluer le taux de détections correctes et la proportion d'objets correctement identifiés parmi ceux présents.
- **Erreur absolue moyenne (MAE) sur le yaw** : mesure l'écart entre l'orientation prédite et l'orientation réelle du véhicule. La perte est pondérée par la valeur absolue de l'angle, ce qui accorde plus d'importance aux erreurs lorsque le yaw est élevé.

- **Erreur sur la vitesse longitudinale (v_x) et latérale (v_y)** : quantifie la précision de la régression des vitesses. La pondération est proportionnelle à l'écart entre la norme de la vitesse et sa valeur moyenne, afin de renforcer l'apprentissage sur les vitesses atypiques.
- **Erreur sur la distance (d)** : évalue la justesse de la prédiction de la distance. Comme pour la vitesse, la pondération dépend de l'écart par rapport à la distance moyenne, ce qui met en avant les cas extrêmes (véhicules très proches ou très éloignés).
- **Temps d'inférence** : mesure le délai nécessaire pour traiter une image, contrainte importante pour garantir le fonctionnement en temps réel sur plateforme embarquée.

La pondération des métriques concernant l'angle, les vitesses et la distance, reflètent le manque de diversité dans les données, nous accordons donc davantage de poids aux cas extrêmes.

À l'issue de ces tests, les modèles présentant le meilleur compromis performance/poids ont été sélectionnés pour la phase finale de développement, en privilégiant précision, faible erreur et temps d'inférence minimal. Ces modèles ont ensuite été intégrés dans une pipeline d'inférence complète, afin d'obtenir de bons résultats.

La dernière partie du développement visait à intégrer cette pipeline ou une partie celle-ci dans le logiciel RTMaps afin de visualiser en temps réel les détections de véhicules. Elle permet de finaliser le projet en concrétisant tout le travail réalisé au préalable.

3.4.2 Tests et validation

Des tests unitaires ont été réalisés sur les modules d'annotation automatique et d'entraînement à l'aide de la bibliothèque `pytest`. Ces tests ont permis de valider :

- L'intégrité des fonctions utilitaires pour les projections caméra et la génération du dataset de détection, et des datasets pour entraîner les modèles de régression
- Le bon déroulement du processus des différents entraînements, en vérifiant la cohérence des entrées/sorties et des formats.
- Le bon fonctionnement du module d'inférence

Cette phase a permis de limiter les régressions et d'assurer la stabilité des modules, garantissant une base logicielle robuste et maintenable.

3.4.3 Difficultés rencontrées

Plusieurs contraintes techniques et organisationnelles ont été identifiées :

- **Ressources matérielles limitées** : la disponibilité, puis la capacité GPU a restreint la vitesse d'entraînement, allongeant certains cycles d'itération.
- **Arrivée tardive des données** : les enregistrements n'ont été reçus qu'en phase avancée, ce qui a nécessité une adaptation rapide des scripts d'annotation et des pipelines.
- **Problèmes de synchronisation Autonoma** : un décalage a été constaté entre les valeurs du GT et les images, désynchronisant légèrement l'image et le GT, et donc leur annotation.

3.5 Résultats obtenus

3.5.1 Présentation des résultats

Annotations automatiques

L'annotation est une étape clé de la pipeline, elle va conditionner la réussite des entraînements. Dans le cadre de ce stage, j'ai travaillé uniquement avec des données simulées, (8000 images, liées chacune à un GT). La suite logique serait d'annoter manuellement quelques données réelles pour compléter notre dataset. Toute annotation automatique sera impossible car les données de vérité terrain ne sont pas disponibles sur les voitures, en temps réel.

Grâce à la réalisation d'un script générique, il est désormais possible de générer automatiquement un dataset complet, structuré en trois *splits* : **train**, **val** et **test**, contenant ces images, annotées avec le véhicule présent dans l'image.

Afin de garantir une compatibilité maximale avec différents modèles, les annotations sont produites simultanément aux formats YOLO, COCO, ainsi que dans un format spécifique attendu par notre réseau CNN. La synchronisation des images avec le GT permet de projeter une bounding box autour de la position théorique du véhicule. Un exemple d'image annotée est présenté ci-dessous :



FIGURE 3.4 – Exemple d'annotation automatique générée par notre script

Cet exemple est fourni uniquement à titre illustratif : en pratique, pour entraîner un modèle, seule l'image brute est conservée, accompagnée de ses annotations dans un fichier séparé. Nous affichons ici l'angle et la distance au véhicule dans un but pédagogique.

Cependant, dans certains cas plus complexes, nous avons observé un décalage entre le GT fourni par le simulateur Autonoma et l'image capturée :



FIGURE 3.5 – Exemple d’annotation présentant un décalage avec l’image

Ce décalage, bien que systématique, diffère d’une image à l’autre, rendant impossible sa correction par l’ajout d’un offset, et entraîne une erreur dans les annotations. Ce phénomène est particulièrement visible dans les virages, ce qui impacte forcément la qualité de l’entraînement.

Dans l’optique de faciliter la visualisation des résultats, une interface Streamlit a été réalisée. Davantage d’informations sont disponibles en annexes : ici.

Annotations corrigées

Comme abordé précédemment, le dataset initial, généré automatiquement, comporte de nombreuses erreurs, de synchronisation notamment. Une manière fiable d’améliorer ces dernières, et d’annoter manuellement les images comportant des erreurs. A l’aide d’un script permettant de récupérer les images avec la plus grande erreur à la prédiction, nous avons pu corriger manuellement près de 800 images, notamment dans les cas complexes comme les virages, où la synchronisation est la moins bonne.

Voici un aperçu du processus de modification, nous pouvons visualiser l’annotation actuelle, avec la prédiction du meilleur modèle actuel en bleu et le GT en rouge. Si ce GT ne nous convient pas, nous pouvons le modifier et créer le rectangle à la main.

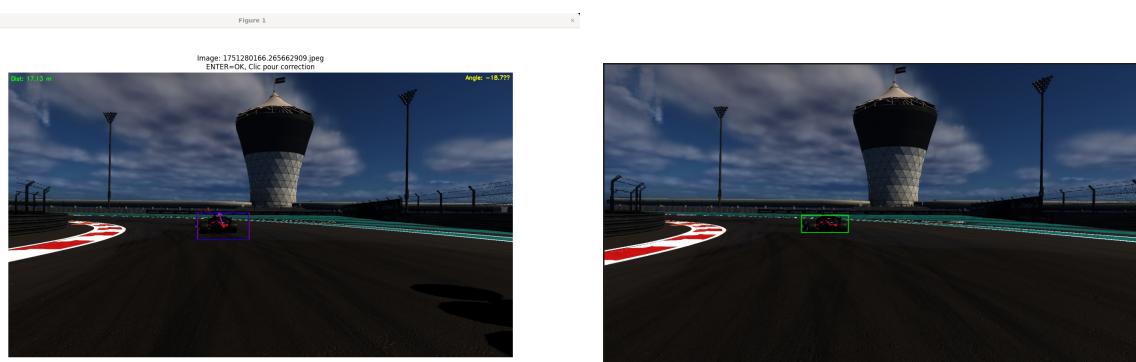


FIGURE 3.6 – Annotation automatique visualisée avec prédiction

FIGURE 3.7 – Annotation manuelle corrigée

Benchmarks de détection

Pour la détection, nous avons concentré nos efforts sur deux familles de modèles : YOLO et RTMDet, en raison de leur légèreté, fiabilité et rapidité, des critères essentiels pour une application en temps réel.

Nous avons d'abord entraîné ces modèles from scratch :

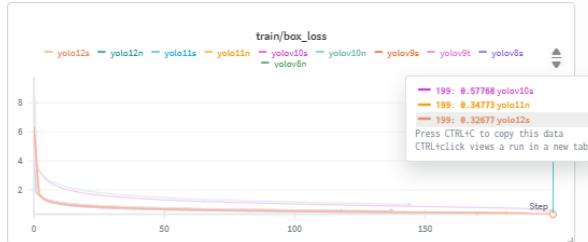


FIGURE 3.8 – Box loss sur le split train des modèles YOLO from scratch

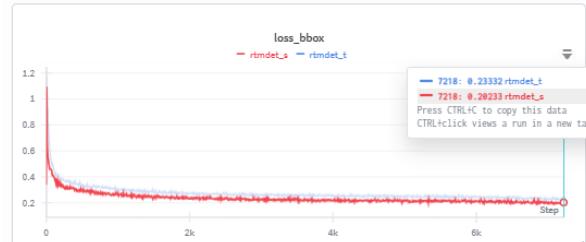


FIGURE 3.9 – Box loss sur le split train des modèles RTMDet from scratch

Après environ 200 époques, la majorité des modèles ont cessé de progresser, et les erreurs restaient élevées pour les modèles YOLO, ce qui illustre la difficulté d'un entraînement *from scratch* sans jeu de données massif. Concernant RTMDet les résultats sont satisfaisants, même meilleurs que ceux de YOLO. A noter cependant que le recall (capacité à détecter toutes les voitures) est très faible pour RTMDet, contrairement à YOLO qui est très proche de 1, pouvant ainsi expliquer cette différence de précision.

Afin d'affiner nos modèles, nous avons réentraîné les mêmes modèles à partir de poids pré-entraînés. Les résultats sont surprenants :

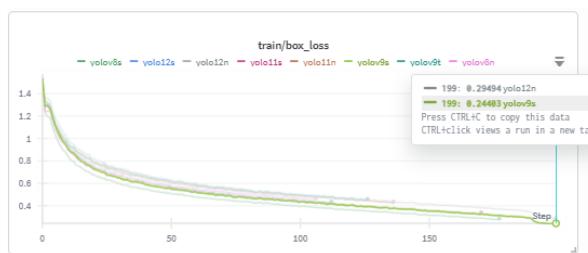


FIGURE 3.10 – Box loss sur le split train des modèles YOLO finetunés



FIGURE 3.11 – Box loss sur le split train des modèles RTMDet finetunés

Nous obtenons alors une précision et un rappel proches de 1, ce qui est idéal : toutes les voitures sont détectées et localisées avec une grande exactitude pour YOLO, tout en ayant des erreurs satisfaisantes.

En ce qui concerne RTMDet, les erreurs ont augmenté par rapport aux modèles précédents, confirmant l'hypothèse d'une détection faible.

Ces premiers entraînements ont été réalisés sur un dataset entièrement généré automatiquement, et reproduit donc les erreurs de celui-ci.

Cependant j'ai aussi pu lancer des entraînements, sur les meilleurs modèles, sur le dataset manuellement corrigé, dans l'optique d'améliorer les résultats dans des situations complexes. Il est à noter que l'introduction d'une partie du dataset corrigé manuellement rend la tâche bien plus complexe, car elles vont différer de celles automatiques, seront moins homogènes et vont introduire une erreur humaine. Il est donc nécessaire d'entraîner davantage les modèles.

Voici alors les résultats obtenus pour 3 des meilleurs modèles, entraînés durant 600 époques :



FIGURE 3.12 – Box loss de 3 des meilleurs modèles sur le dataset corrigé

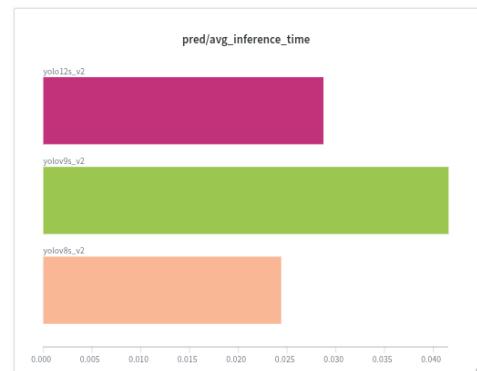
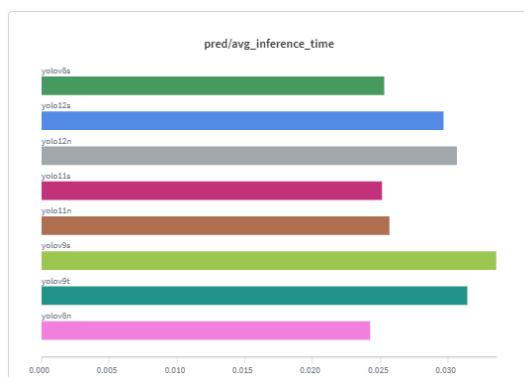


FIGURE 3.13 – Temps d'inférence moyen par frame de 3 des meilleurs modèles sur le dataset corrigé

Les modèles doivent fonctionner en temps réel. Nous avons donc testé uniquement les variantes **nano** et **small**, offrant un compromis optimal entre précision et rapidité :



(a) Temps d'inférence moyen par frame des modèles YOLO finetunés



(b) Temps d'inférence moyen par frame des modèles RTMDet finetunés

Les meilleurs modèles atteignent 20 ms par image (environ 50 FPS), ce qui est compatible avec des applications embarquées. Cependant, même si les résultats sont très bons, cela signifie que le modèle reproduit de manière très précise ce qui lui a été donné en tant qu'annotations, ce qui veut dire que toute erreur d'annotation se répercute aussi dans le modèle.

Davantage de résultats sont disponibles en annexes : ici.

Benchmarks de prédiction d'angle, de vitesse et de distance

L'objectif des modèles de régression est de prédire le *yaw* (orientation), les vitesses longitudinale v_x et latérale v_y , ainsi que la distance tout en restant très rapides.

Nous avons testé plusieurs architectures, avec des poids pré-entraînés :

- **Image croppée** : Pour chacune des détections provenant du modèle de détection, nous isolons uniquement la bbox de l'image afin d'estimer l'angle, la vitesse et la distance uniquement à partir du véhicule.
- **Image croppée + bbox** : Pour chacune des détections, nous fournissons au modèle à la fois l'image croppée, avec un léger padding, et la bounding box normalisée de la détection, censée correspondre à l'emplacement exact du véhicule.



FIGURE 3.15 – Total loss (yaw + vitesses + distance) sur le split, en entraînant avec l'image croppée à la bbox uniquement



FIGURE 3.16 – Total loss (yaw + vitesses + distance) sur le split, en entraînant avec l'image croppée et la bbox normalisée

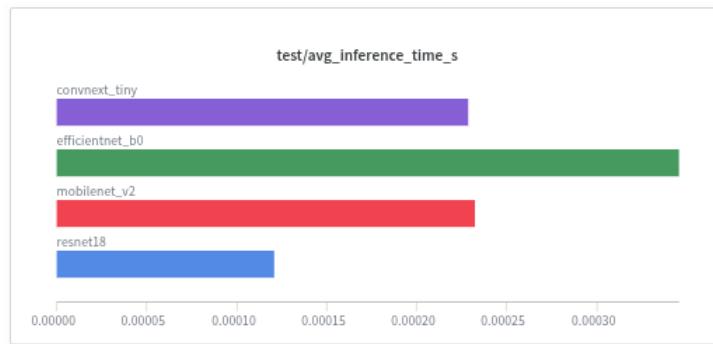


FIGURE 3.17 – Temps d'inférence moyen par frame des modèles CNN finetunés

L'ajout de la prédiction du *yaw*, des vitesses v_x , v_y et de la distance permet de fournir une détection enrichie, adaptée au contexte de course automobile, tout en respectant les contraintes de temps réel. Cependant sa mise en oeuvre dépend grandement de la diversité des annotations, et de la qualité du dataset. Malheureusement, dans notre cas, les données fournies ne permettent pas un entraînement de qualité et fiable, des biais sont introduits à l'entraînement car les valeurs ne fluctuent que très rarement. Lors de la prédiction, les valeurs sont donc arbitraires, et plutôt moyennes, présentant alors de grandes erreurs en prédiction, et sans prédire aucune dynamique.

Davantages de résultats sont disponibles en annexes : ici.

Module d'inférence complet

Une fois le modèle de détection et le second modèle de régression (prédition de l'angle de lacet, de la vitesse et de la distance) finalisés, il devient nécessaire de les combiner au sein d'un même module. L'objectif est de fournir, pour une image donnée, non seulement les résultats de détection (présence et position des véhicules), mais aussi des informations dynamiques comme leur orientation (*yaw*), leur vitesse et leur distance estimée.

Ce module doit être conçu de manière suffisamment générique pour permettre une utilisation souple : sa structure doit faciliter les évolutions futures, qu'il s'agisse de remplacer un modèle par une version plus performante, ou d'ajouter de nouvelles métriques (accélération, trajectoire prédictive, etc.).

Dans l'optique de visualiser directement les résultats sur des images ou des séquences, une interface interactive a également été développée avec la technologie Streamlit. Cette interface, permet d'explorer facilement les sorties des modèles : affichage des boîtes englobantes, orientation estimée, vecteurs de vitesse et distances projetées, le tout superposé sur l'image d'entrée. Elle constitue ainsi un outil pratique pour la validation visuelle et l'expérimentation rapide.

Davantages d'informations sont disponibles en annexes : ici.

Module RTMaps

Un des objectifs principaux abordés lors de l'analyse du cahier des charges, est la création du module RTMaps, afin d'intégrer cette pipeline en temps réel dans le véhicule autonome.

Le résultat final correspond à ce diagramme :

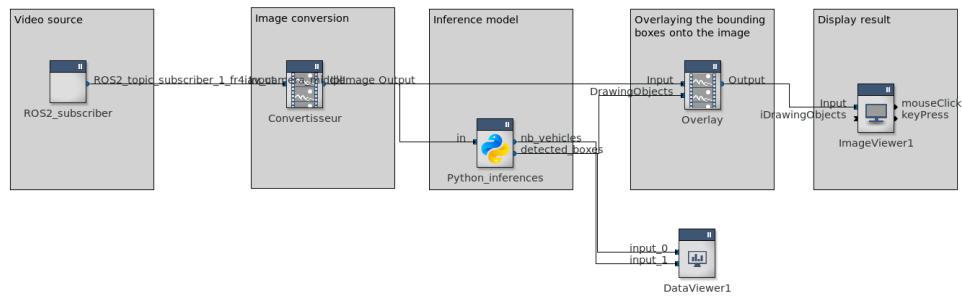


FIGURE 3.18 – Diagramme RTMaps en temps réel

Ce diagramme récupère en entrée une source vidéo : ici le module est abonné à un topic provenant d'un fichier rosbag qui contient un enregistrement de toutes les données du simulateur Autonomia dont la caméra, et récupèrera les images par ce biais, mais cela sera remplacé par un composant caméra, correspondant aux images reçues par le véhicule.

Cette image reçue est alors convertie dans un format propre à RTMaps afin d'être manipulée : IPLImage. Pour chacune d'entre elles, une inférence est réalisée sur nos modèles via le composant Python qui est notre module d'inférence présenté précédemment. Nous récupérons les voitures détectées, chacune associée à un angle une vitesse et une distance, et le nombre de véhicules. Le composant Overlay et ImageViewer permettent à l'homme de visualiser ces résultats directement en temps réel, mais ils ne sont pas nécessaires au bon fonctionnement du diagramme.

Ce diagramme est générique, il est facilement possible de changer les modèles utilisés, et les entrées vidéos aussi, permettant une maintenance et une intégration efficace dans le véhicule.

Après avoir choisi sur quels modèles inférer, lors de l'exécution, il est alors possible de visualiser en temps réel la détection du véhicule :

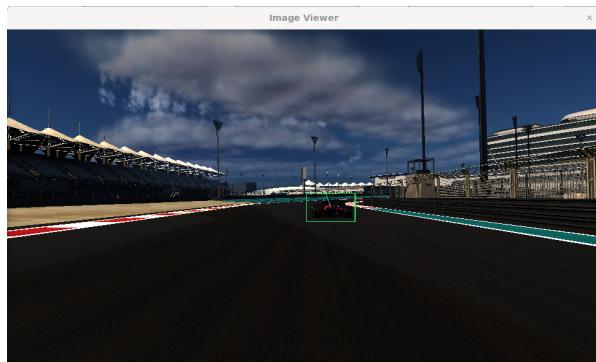


FIGURE 3.19 – Image en sortie de diagramme temps réel

	Rectangle
[0]	{73.184,53}
Id	0
Color	3
Width	696
x1	525
y1	898
x2	624
y2	Y.-35.84 VX:6.44 VY:-4.88 D:19.8
[1]	
[2]	
Id	0
Color	0
Width	0
x	0
y	0
char width	0
char height	0
orientation	0
background color	{0,0,0}
text	
[3]	
[4]	
[5]	
[6]	
[7]	
[8]	
[9]	
[10]	

FIGURE 3.20 – Données en sortie de diagramme temps réel

Nous pouvons constater visuellement la mauvaise prédiction de l'angle, et obtenons toutes les informations nécessaires : les 4 points de la détection 2D, ainsi qu'un champ *text* dans lequel nous retrouvons les valeurs numériques de l'angle (Y), la vitesse relative (VX et VY) et une distance (D).

Davantages de précisions techniques sont disponibles en annexes : ici.

3.5.2 Analyse critique

Si les résultats sont encourageants, certains cas complexes (scénarios de dépassement, présence de plusieurs véhicules à détecter simultanément) restent problématiques, car ces configurations n'ont pas pu être testées faute de données correspondantes. La robustesse en conditions réelles reste donc à valider lors d'essais sur piste.

De plus il ne faut pas oublier que les annotations sont bonnes, mais pas parfaites. L'automatisation de celles-ci permet de réaliser un grand volume de données rapidement, mais introduit des erreurs, notamment par le simulateur Autonoma.

3.5.3 Axes d'amélioration

Plusieurs pistes d'amélioration ont été abordées et peuvent être envisagées pour renforcer la robustesse et la performance de la solution développée :

- **Enrichissement des données d'entraînement :** Les données utilisées proviennent uniquement de la caméra centrale, ne comportent qu'un seul véhicule détecté par image, et très peu lors de virages. Cette faible diversité limite la capacité de généralisation du modèle et introduit un manque de fiabilité dans ces cas précis. Il serait pertinent d'intégrer des données issues de caméras supplémentaires (latérales, arrière) et d'inclure des scénarios multi-véhicules afin de mieux représenter les conditions réelles de course. Bien que des données latérales aient été disponibles, l'absence de détections les a rendues inexploitables. A cela s'ajoute énormément de redondance, rendant très complexe les tâches de prédiction d'angle, de vitesse et de distance. Enfin, il est indispensable de valider en continu la qualité des annotations générées, et le cas échéant de corriger certaines d'entre elles, afin de garantir un jeu d'entraînement fiable et représentatif.
- **Entraînement et fine-tuning sur données réelles :** Les données synthétiques issues du simulateur, bien que précises, ne reproduisent pas parfaitement toutes les variations visuelles et dynamiques rencontrées sur piste. Un entraînement complémentaire ou un fine-tuning sur des données réelles annotées manuellement permettrait d'améliorer la robustesse en conditions opérationnelles.
- **Intégration de données multi-capteurs :** L'ajout de données provenant d'autres capteurs (LiDAR, radar, IMU) pourrait compléter les informations visuelles, renforcer la perception globale et améliorer la détection et le suivi dans des conditions difficiles, comme par mauvais temps ou faible luminosité.

Ces axes ciblent la qualité et la diversité des données, et la richesse des sources d'information, afin d'adapter la solution aux exigences du monde réel.

Chapitre 4

Développement Durable et Responsabilité Sociétale

Sommaire

4.1	Impacts environnementaux	33
4.2	Aspects sociaux et éthiques	33
4.3	Bonnes pratiques mises en œuvre	33

4.1 Impacts environnementaux

Le projet A2RL comporte plusieurs dimensions environnementales. L'optimisation de la conduite des véhicules autonomes — par une meilleure gestion de la vitesse, une anticipation accrue et la réduction des freinages brusques — contribue directement à limiter la consommation de carburant et les émissions de CO₂. D'autres aspects méritent également d'être pris en compte :

- **Consommation énergétique des outils** : l'entraînement des modèles de détection repose sur des ressources informatiques lourdes (GPU, CPU), entraînant une consommation électrique notable. D'où l'importance d'optimiser les architectures et les calculs.
- **Choix du matériel** : la sélection de capteurs (caméras, LiDAR) plus durables et moins énergivores, ainsi que leur longévité, influencent fortement l'empreinte globale du projet.
- **Cycle de vie logiciel** : une maintenance régulière, la réutilisation du code et l'amélioration continue limitent le besoin de recalibrage matériel et évitent le gaspillage associé.

4.2 Aspects sociaux et éthiques

Au-delà de la technique, le projet soulève des enjeux humains. La sécurité reste la priorité : il est indispensable de clarifier les responsabilités en cas d'accident ou de défaillance, et d'assurer une transparence sur les limites des algorithmes. En parallèle, la course autonome représente un moyen de diminuer considérablement les risques liés à l'erreur humaine, qui demeure la principale cause d'accidents. Elle peut donc jouer un rôle clé dans la protection des pilotes.

4.3 Bonnes pratiques mises en œuvre

Pour garantir la qualité du développement, le projet s'appuie sur plusieurs pratiques éprouvées. Le travail avance de manière itérative, avec des revues hebdomadaires permettant d'intégrer rapidement les retours et d'améliorer en continu. La gestion de version est assurée via Git, garantissant une traçabilité claire des évolutions et facilitant la collaboration entre les futurs membres de l'équipe. Enfin, des tests unitaires sont disponibles et une documentation complète, régulièrement mise à jour, accompagne le projet afin de simplifier la maintenance et la transmission des connaissances.

Chapitre 5

Conclusion

Sommaire

5.1 Bilan du stage	35
5.2 Évolutions possibles	35

5.1 Bilan du stage

Ce stage a été une expérience particulièrement formatrice, tant sur le plan technique qu'humain et professionnel. L'objectif principal — concevoir et mettre en place une chaîne de perception basée sur la vision — a été atteint. Les modules développés ont donné des résultats concluants en simulation et constituent une base solide pour les futurs essais en conditions réelles.

Parmi les points forts, on peut souligner la mise en œuvre d'architectures modernes pour la détection d'objets et la régression de paramètres dynamiques, la création d'un pipeline générique et reproductible pour l'annotation, l'entraînement et l'inférence, ainsi que l'intégration du travail dans un environnement RTMaps, permettant d'évaluer rapidement les performances en temps réel. Toute l'architecture étant réalisée, le reste du travail porte uniquement sur l'amélioration des premiers travaux.

Certaines difficultés ont également été identifiées, notamment la dépendance aux données disponibles, encore limitées en volume et diversité, la contrainte temps réel nécessitant une optimisation importante des algorithmes, et la complexité de l'intégration dans une architecture logicielle et matérielle complète, qui requiert une coordination avec plusieurs équipes.

Cette expérience m'a permis d'approfondir mes compétences en apprentissage profond appliqué à la vision par ordinateur, de découvrir concrètement les enjeux industriels des véhicules autonomes, et de développer mes capacités à collaborer dans un environnement pluridisciplinaire tout en respectant des contraintes de temps et d'objectifs. L'ensemble constitue une base solide pour poursuivre mon évolution dans ce domaine en pleine expansion.

5.2 Évolutions possibles

Plusieurs pistes peuvent être envisagées après la fin de ce stage. La prochaine étape majeure consiste à tester le système lors d'essais en conditions réelles, prévus en octobre à Abu Dhabi, afin de valider ses performances dans un environnement dynamique. Pour améliorer la robustesse et la précision des modèles, il sera également nécessaire de concevoir des entraînements plus sophistiqués, intégrant des données plus variées, des scénarios multi-véhicules et des augmentations de données avancées, afin d'assurer de meilleurs résultats de détection, mais aussi et surtout sur la partie régression. Enfin, la poursuite de l'optimisation des algorithmes pour répondre aux contraintes temps réel et l'intégration des modules dans la chaîne complète du véhicule, ainsi que l'extension des capacités de perception par fusion avec d'autres capteurs comme le LiDAR, sont des évolutions clés pour la suite du projet.

Ce stage fera également l'objet d'une reprise dans le cadre d'un cours à l'INSA dès la rentrée. Le projet servira alors de support pédagogique pour les étudiants, qui pourront s'appuyer sur les résultats obtenus afin de prolonger et d'enrichir le travail amorcé. Cette continuité ouvrira la voie à de nombreuses pistes d'amélioration, qu'il s'agisse d'explorer d'autres architectures de détection, de diversifier les scénarios de test, d'intégrer de nouvelles sources de données ou encore d'optimiser l'intégration temps réel. Ainsi, le stage constitue non seulement une première étape concrète dans le développement d'une solution de perception embarquée, mais également une base de travail évolutive destinée à être approfondie dans un cadre académique.

Chapitre 6

Bibliographie

Sommaire

6.1 Bibliographie	37
--------------------------------	----

6.1 Bibliographie

- [1] Ultralytics Hyperparameters, *Hyperparameter Tuning - Custom Search Space Example*, 2024.
- [2] ResearchGate, *Basic architecture of the YOLO series network presented as backbone, neck and detect head*, 2023.
- [3] Ultralytics Benchmark, *Model Benchmarking with Ultralytics YOLO*, 2024.
- [4] Malla, S. ; Kuo, C.-C. J. ; Chen, C.-Y. ; et al., *Object Detection in Adverse Weather for Autonomous Driving through Data Merging and YOLOv8*, Sensors, vol. 23, no. 20, art. 8471, 2023.
- [5] Replication Study and Benchmarking of Real-Time Object Detection Models, *Cornell University*, 2024.
- [6] MMDetection Github, *open-mmlab/mmdetection*, 2019.
- [7] Ultralytics Github, *ultralytics*, 2025.
- [8] Hugging Face Objet Detection Leaderboard, *object-detection-leaderboard*, September, 7th 2023.
- [9] Benchmarking Deep Learning Models for Object Detection on Edge Computing Devices, Daghsh K. Alqahtani, Aamir Cheema, Adel N. Toosi *arxiv.org*, September, 25th 2024.
- [10] In-Depth Analysis of the Best Object Detection Models for 2024, Allan Kouidri *top-object-detection-models-review*, September, 2024.
- [11] RTMaps™ Middleware *rtmaps*, 2024.

Chapitre 7

Annexes

Sommaire

7.1 Benchmarks	39
7.1.1 Benchmarks complets des modèles YOLO	39
7.1.2 Benchmarks complets des modèles RTMDet	48
7.1.3 Benchmarks de modèle angle, vitesse et distance	52
7.2 RTMaps	56
7.3 Interface Streamlit	61
7.3.1 Application d'annotation	61
7.3.2 Application d'inférence	63

7.1 Benchmarks

7.1.1 Benchmarks complets des modèles YOLO

Modèles finetunés

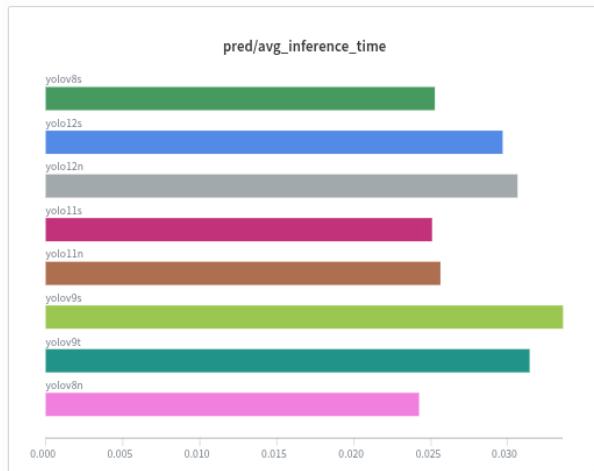


FIGURE 7.1 – Inference times



FIGURE 7.2 – Train / Box loss

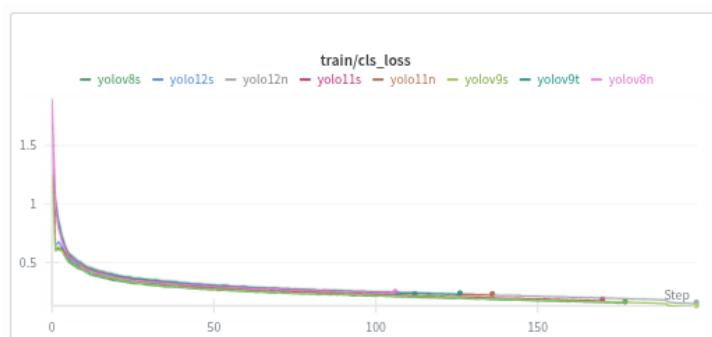


FIGURE 7.3 – Train / Cls loss

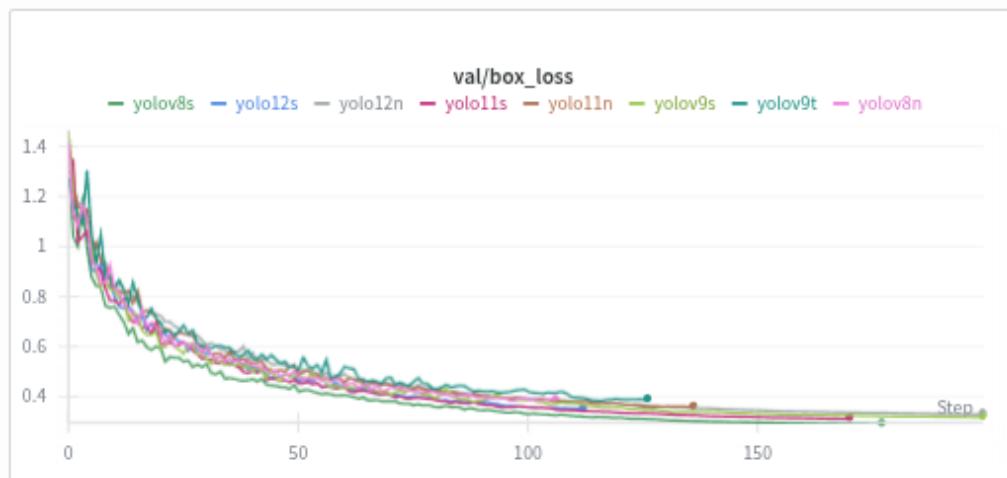


FIGURE 7.4 – Val / Box loss

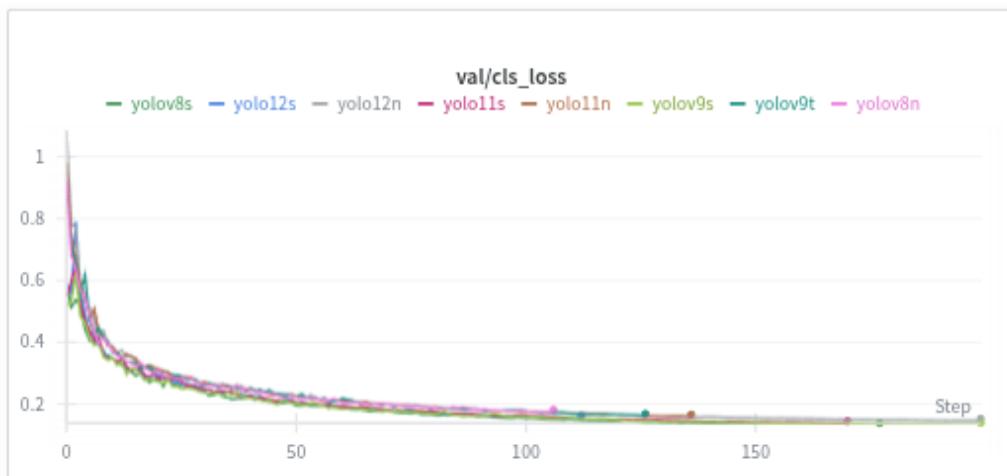


FIGURE 7.5 – Val / Cls loss

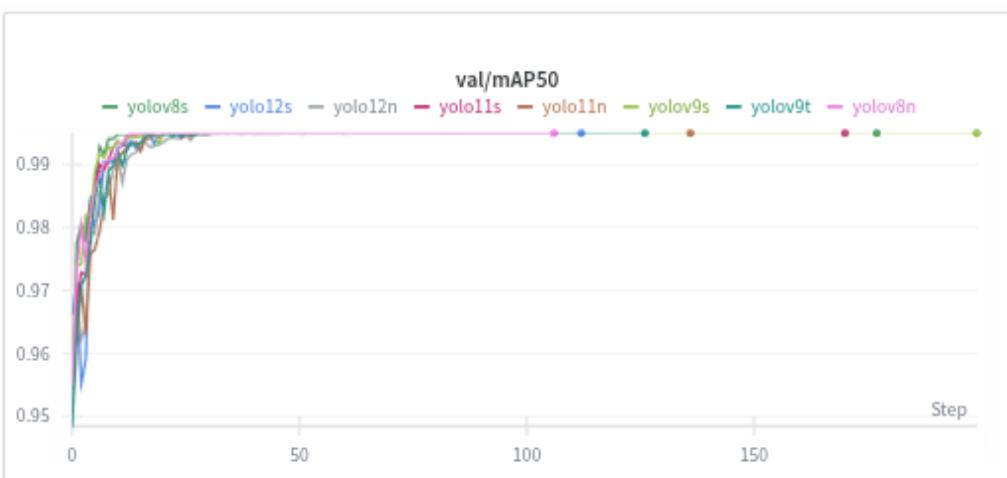


FIGURE 7.6 – Val / MAP50

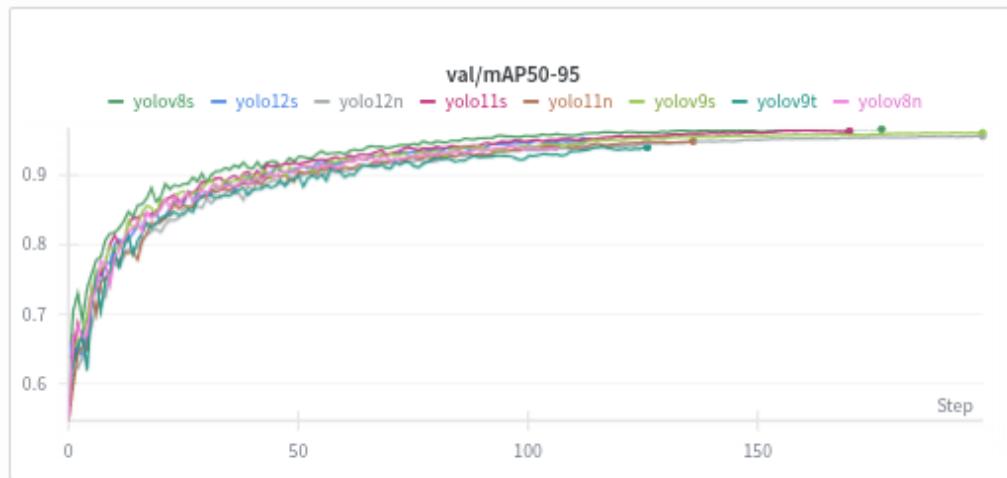


FIGURE 7.7 – Val / MAP50-95

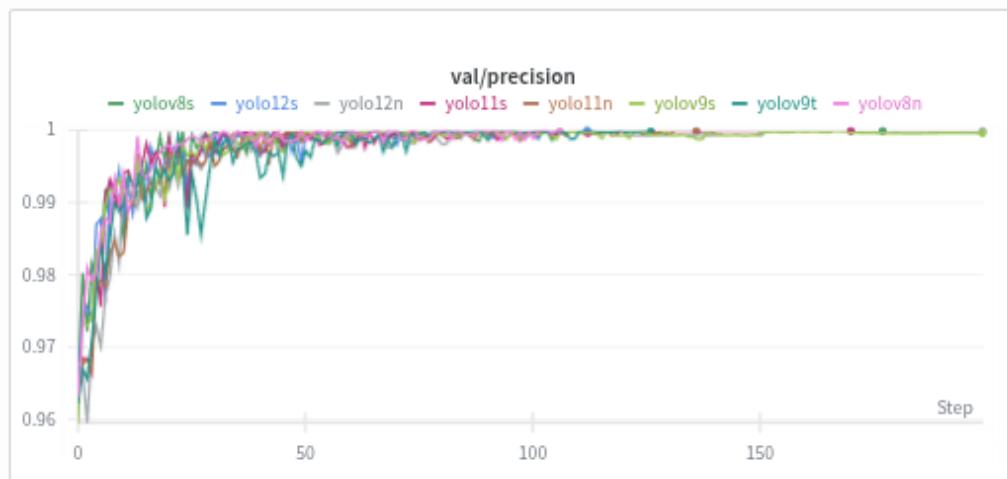


FIGURE 7.8 – Val / Precision

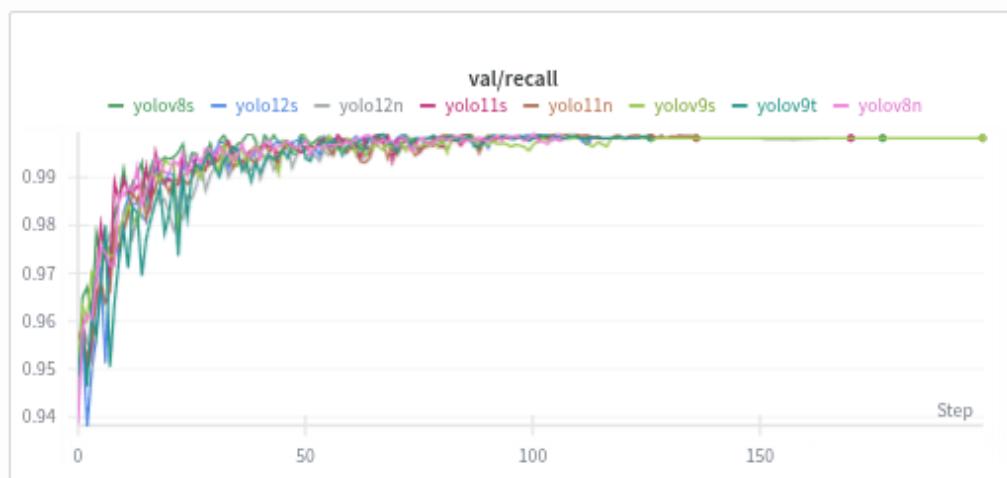


FIGURE 7.9 – Val / Recall

Modèles entraînés from scratch

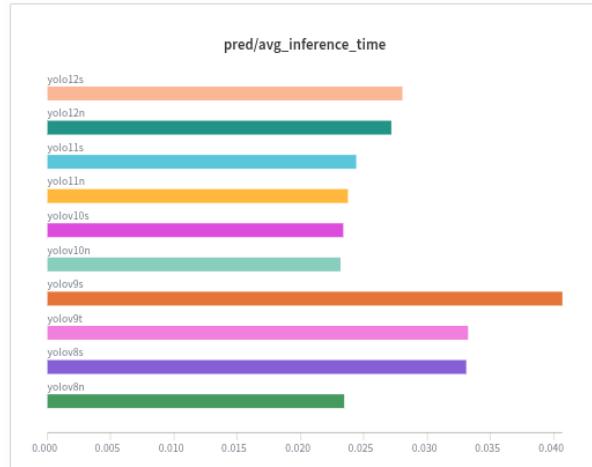


FIGURE 7.10 – Inference times

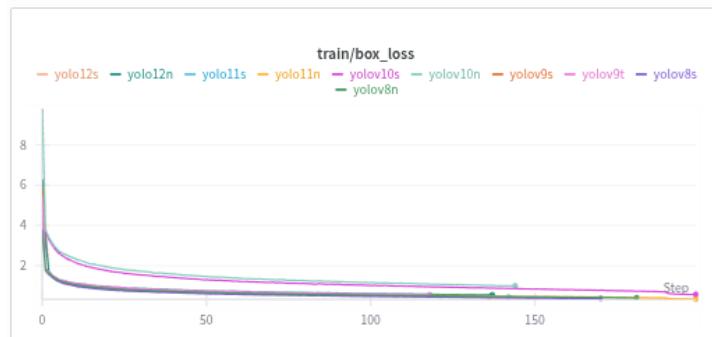


FIGURE 7.11 – Train / Box loss

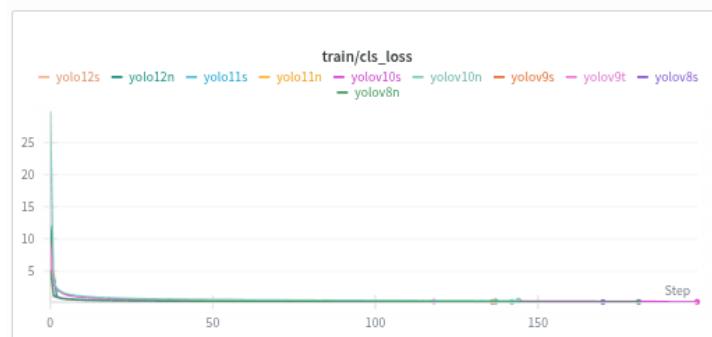


FIGURE 7.12 – Train / Cls loss

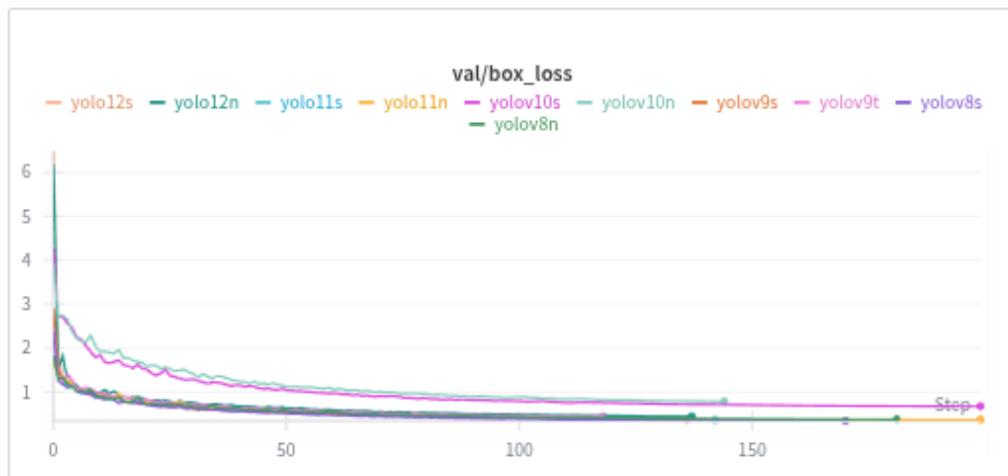


FIGURE 7.13 – Val / Box loss

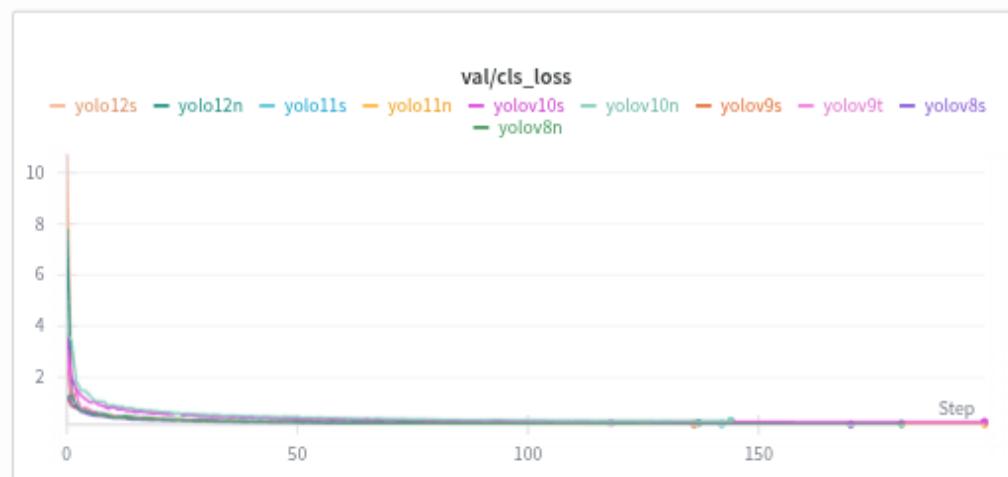


FIGURE 7.14 – Val / Cls loss

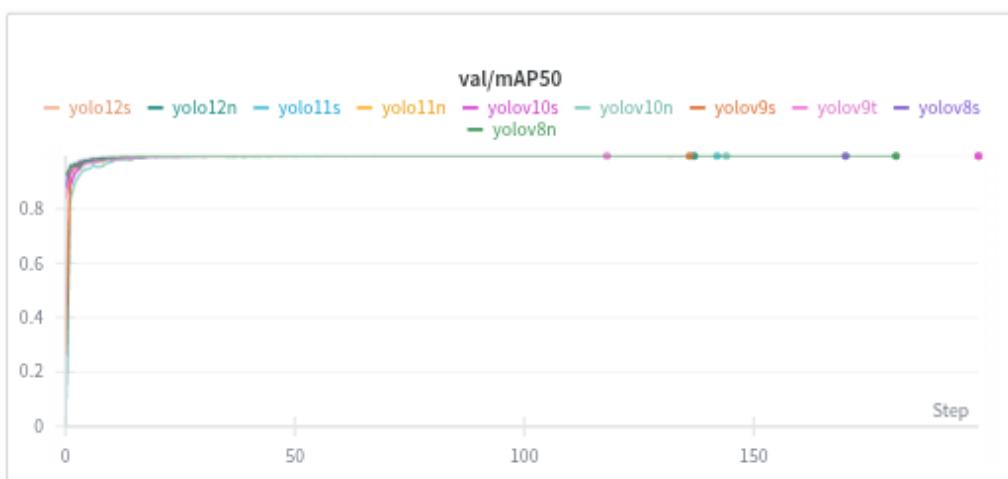


FIGURE 7.15 – Val / MAP50

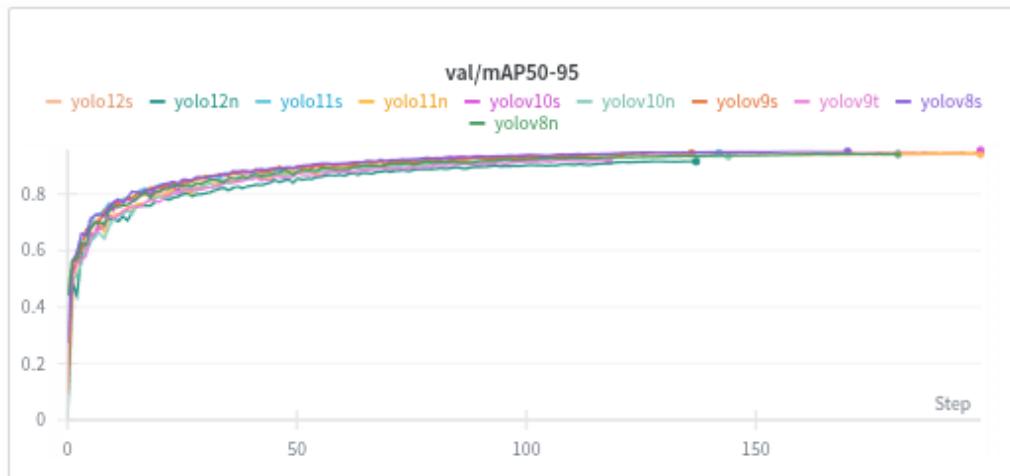


FIGURE 7.16 – Val / MAP50-95

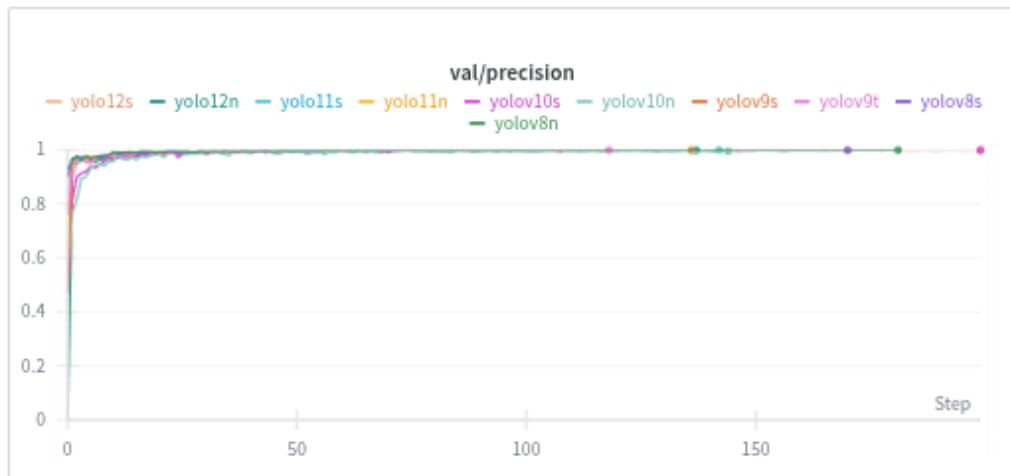


FIGURE 7.17 – Val / Precision

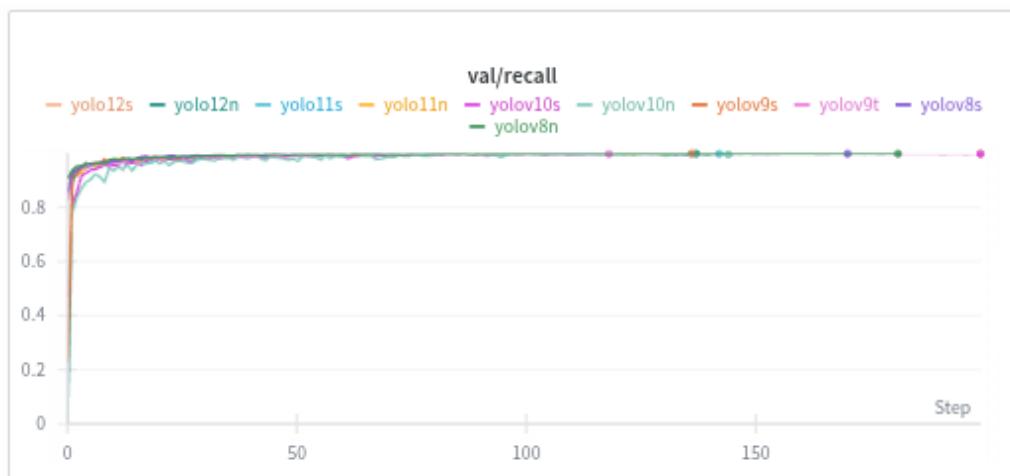


FIGURE 7.18 – Val / Recall

Entraînement sur annotations corrigées



FIGURE 7.19 – Inference times

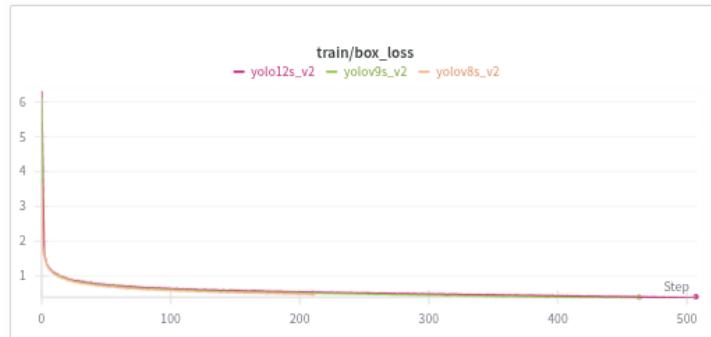


FIGURE 7.20 – Train / Box loss

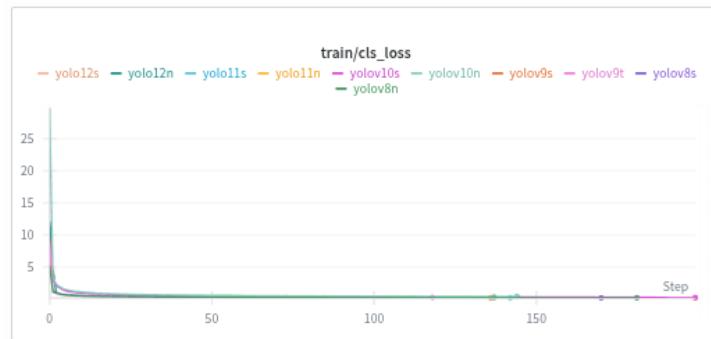


FIGURE 7.21 – Train / Cls loss

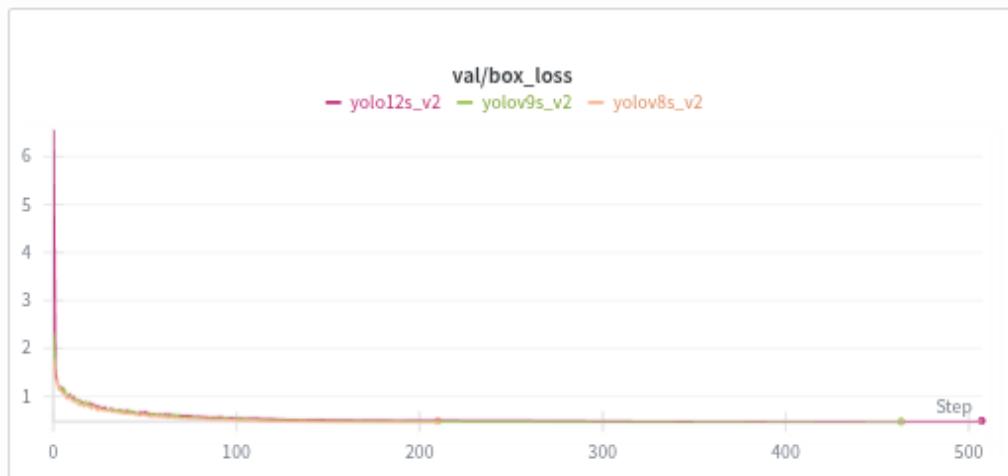


FIGURE 7.22 – Val / Box loss

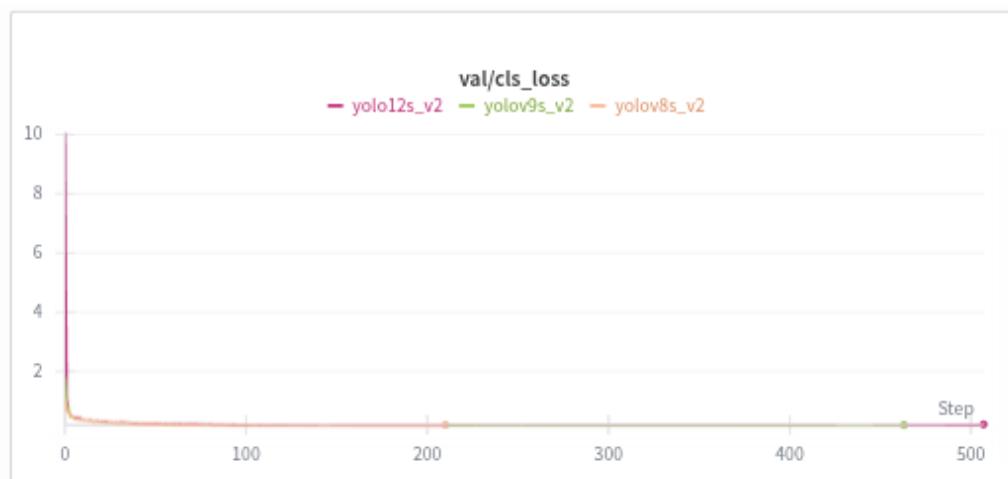


FIGURE 7.23 – Val / Cls loss



FIGURE 7.24 – Val / MAP50

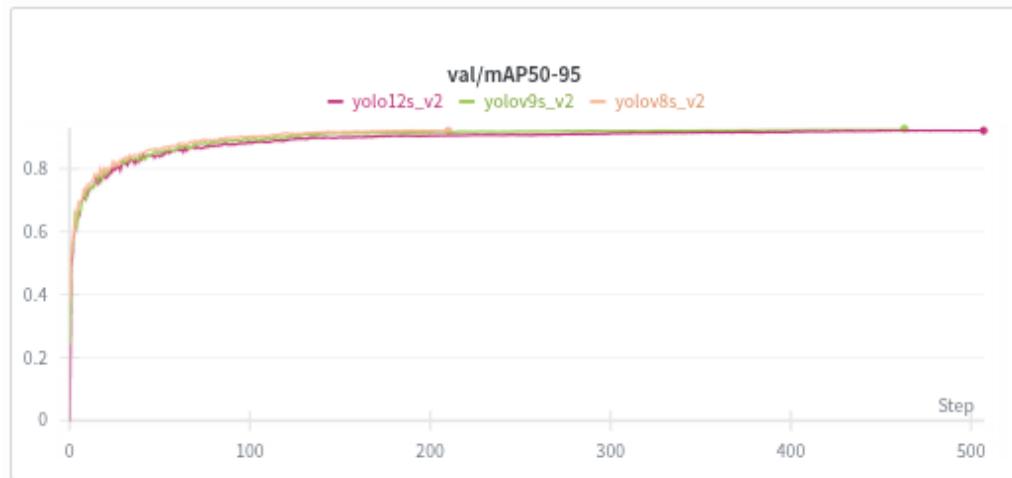


FIGURE 7.25 – Val / MAP50-95



FIGURE 7.26 – Val / Precision

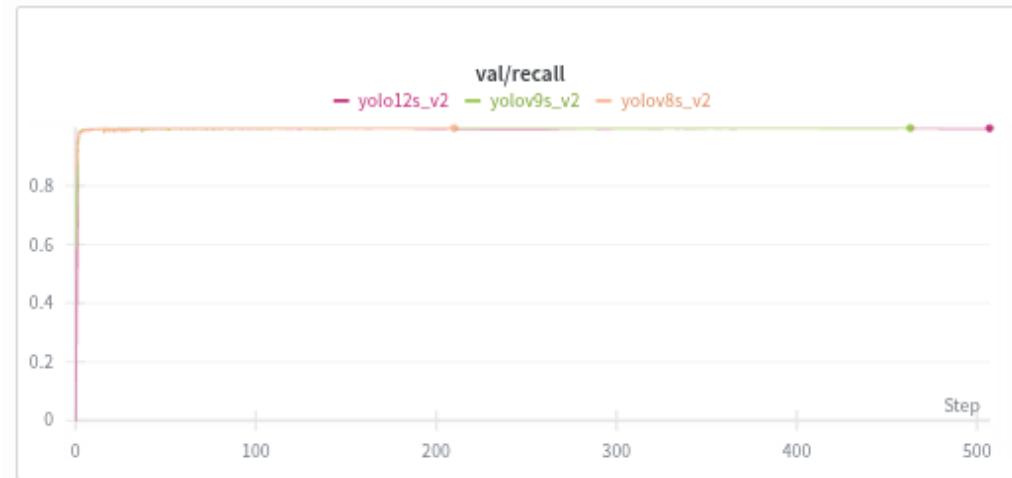


FIGURE 7.27 – Val / Recall

7.1.2 Benchmarks complets des modèles RTMDet Modèles finetunés



FIGURE 7.28 – Inference times

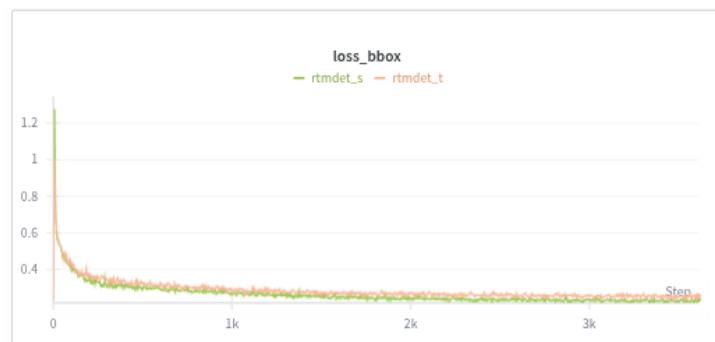


FIGURE 7.29 – Train / Box loss

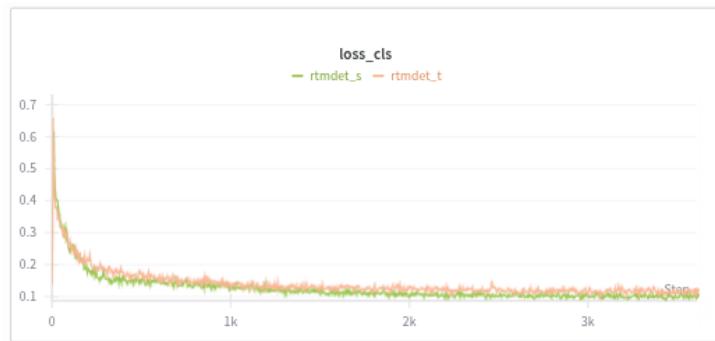


FIGURE 7.30 – Train / Cls loss

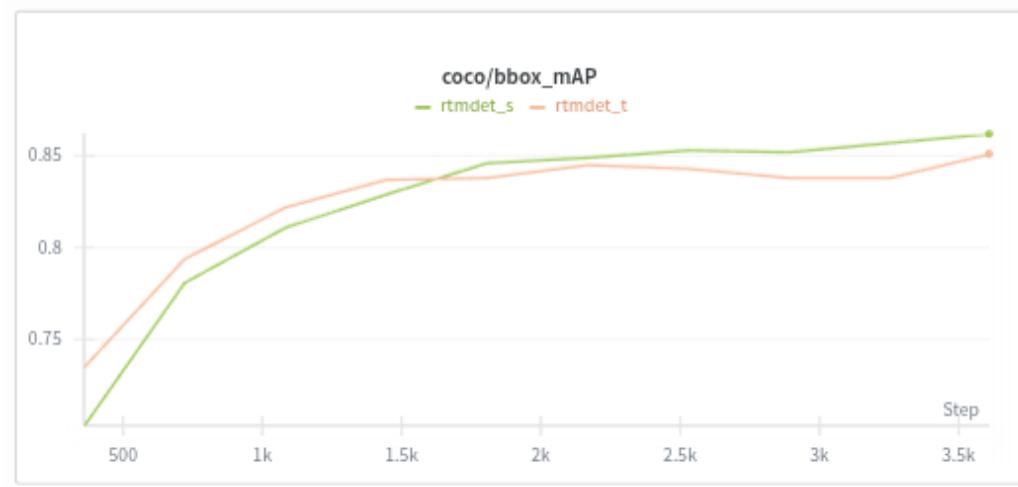


FIGURE 7.31 – Val / MAP

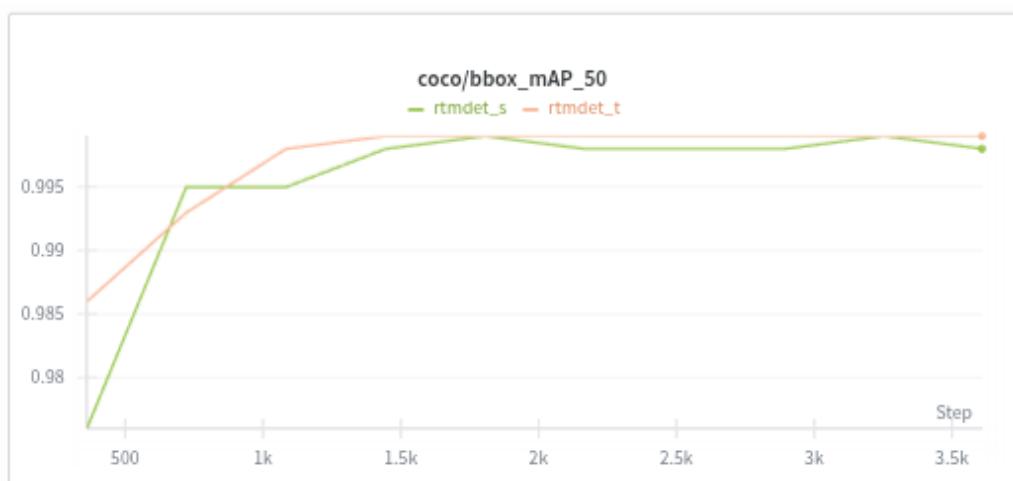


FIGURE 7.32 – Val / MAP50

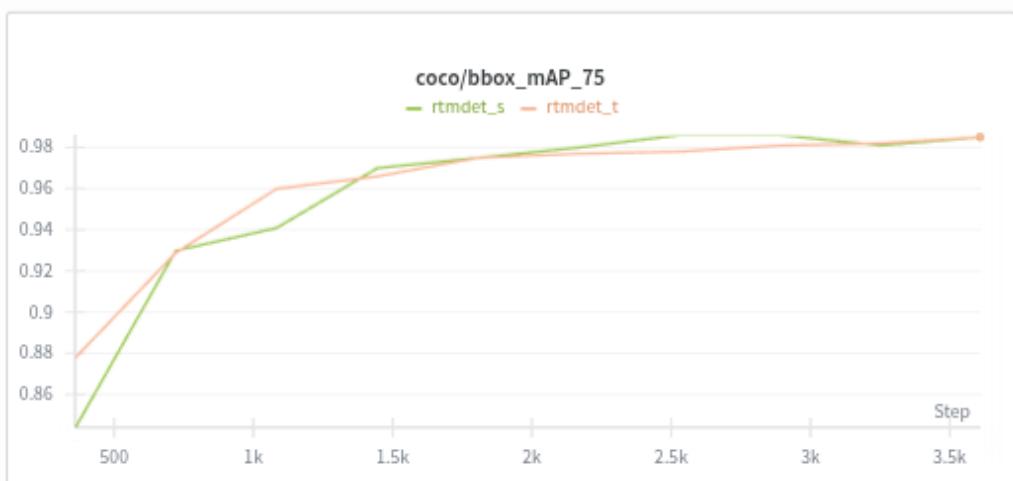


FIGURE 7.33 – Val / MAP75

Modèles from scratch

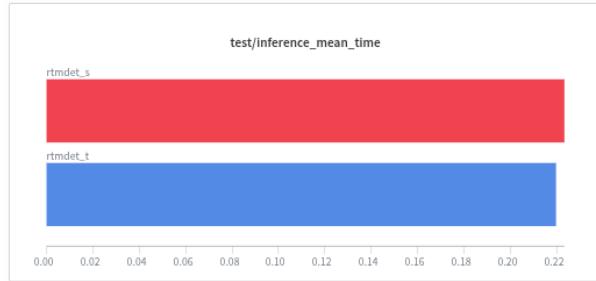


FIGURE 7.34 – Inference times

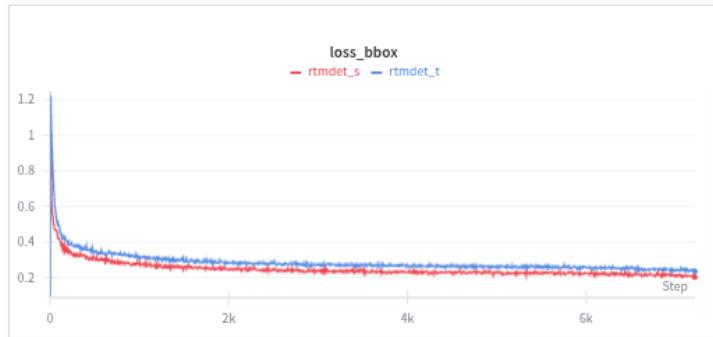


FIGURE 7.35 – Train / Box loss

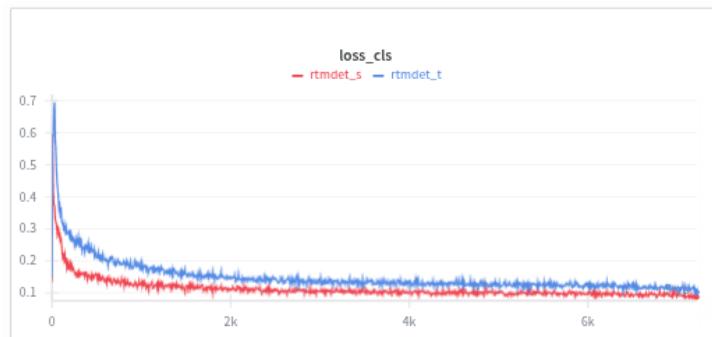


FIGURE 7.36 – Train / Cls loss

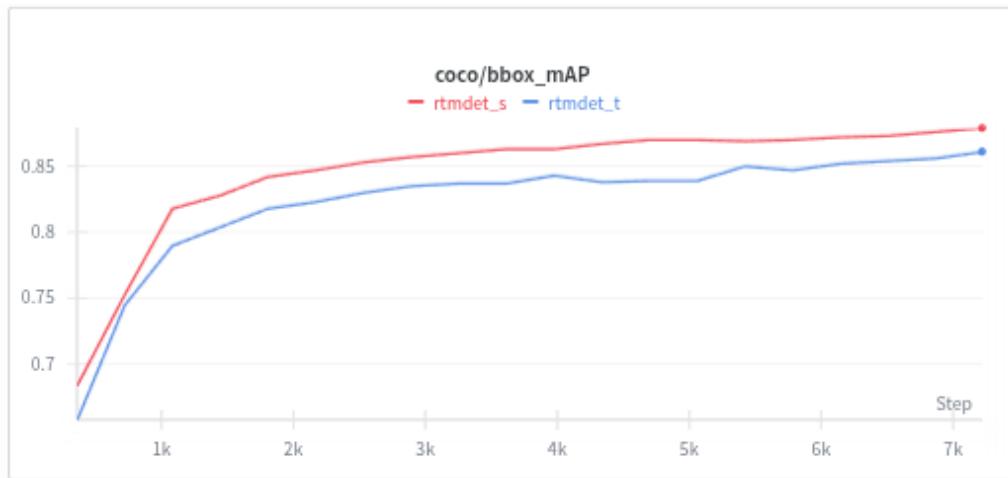


FIGURE 7.37 – Val / MAP

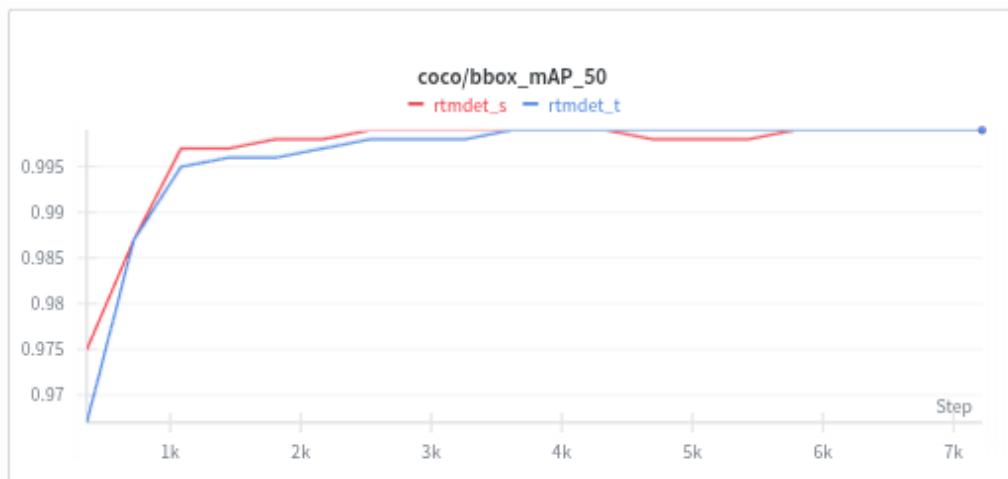


FIGURE 7.38 – Val / MAP50

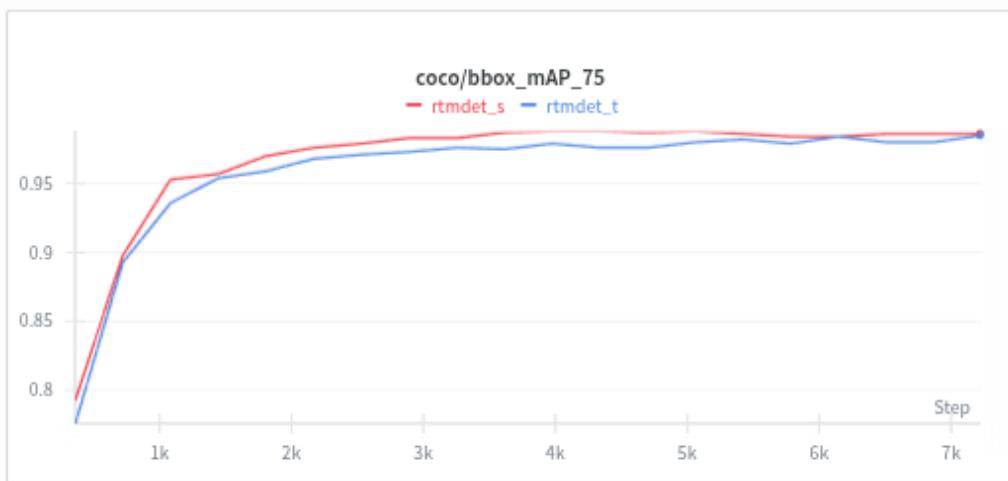


FIGURE 7.39 – Val / MAP75

7.1.3 Benchmarks de modèle angle, vitesse et distance

Architecture Image Cropped

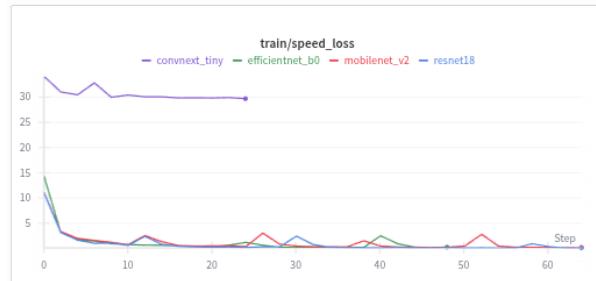


FIGURE 7.40 – Train / Speed Loss

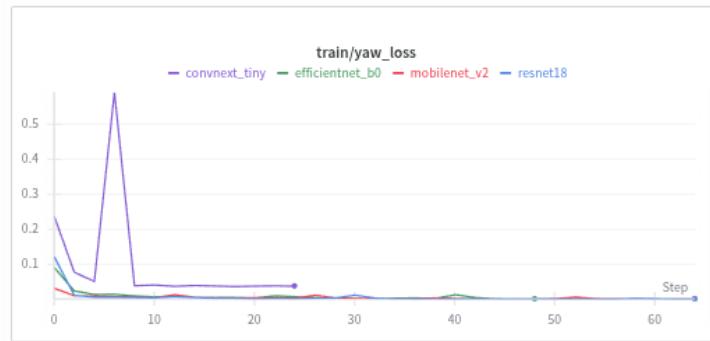


FIGURE 7.41 – Train / Yaw Loss

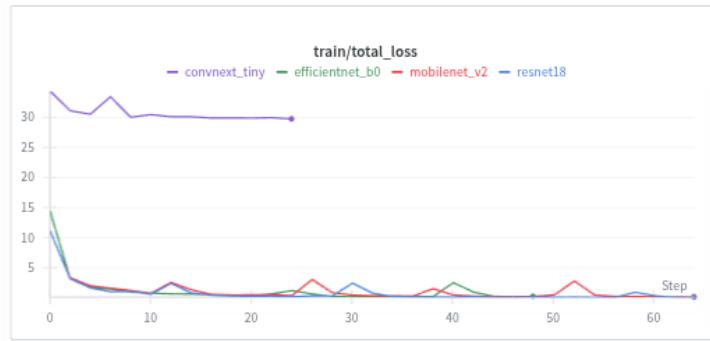


FIGURE 7.42 – Train / Total Loss

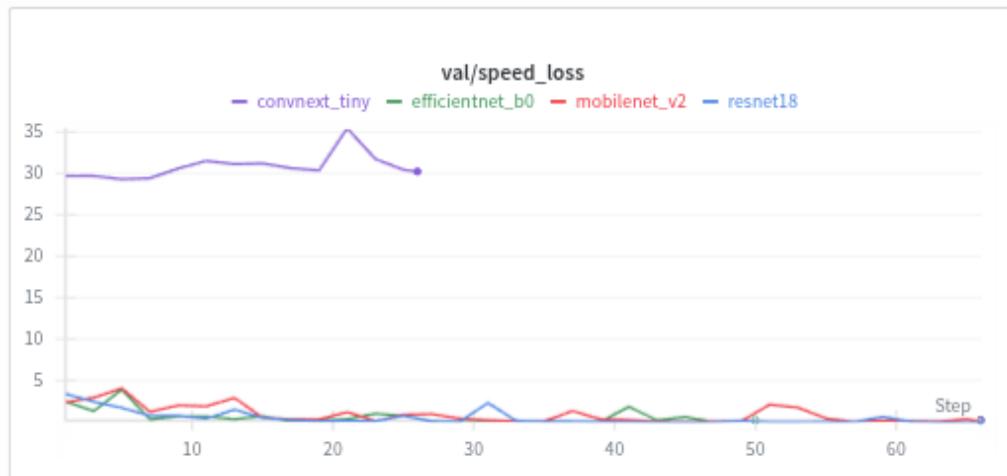


FIGURE 7.43 – Val / Speed Loss

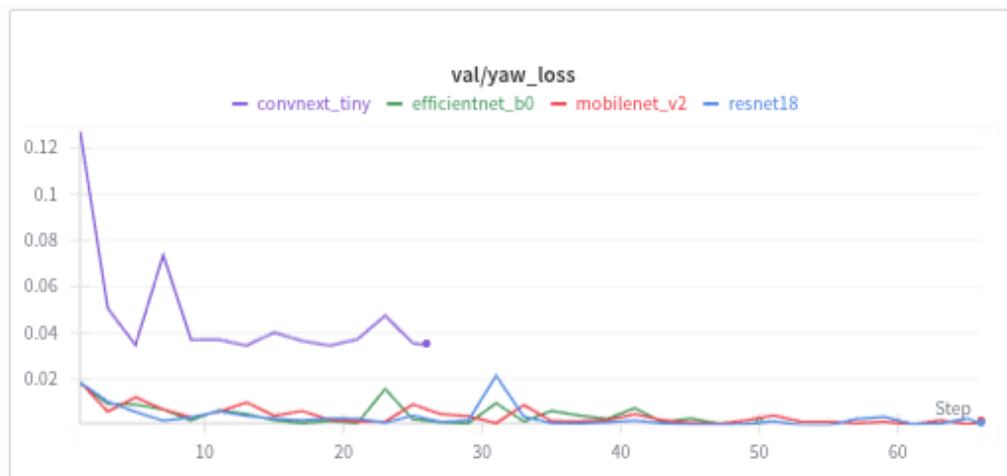


FIGURE 7.44 – Val / Yaw Loss

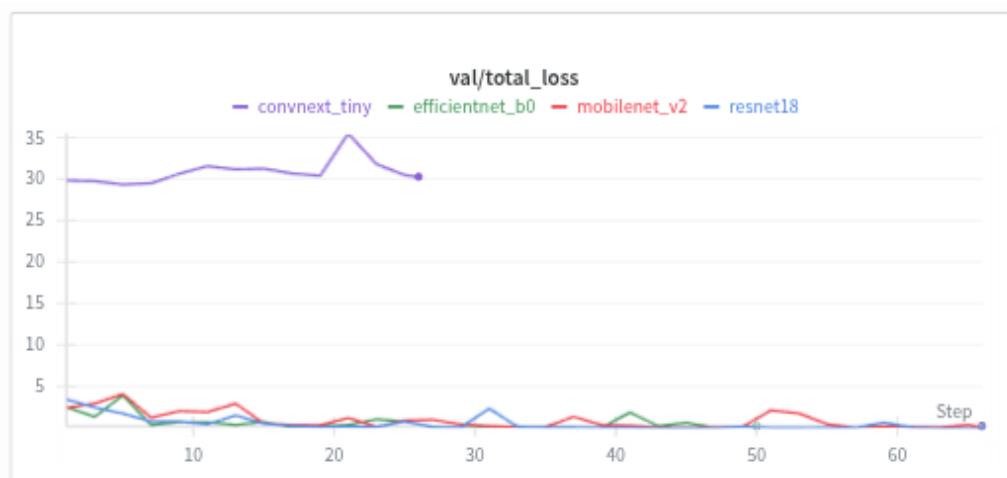


FIGURE 7.45 – Val / Total Loss

Architecture Image + Bounding box



FIGURE 7.46 – Train / Speed Loss

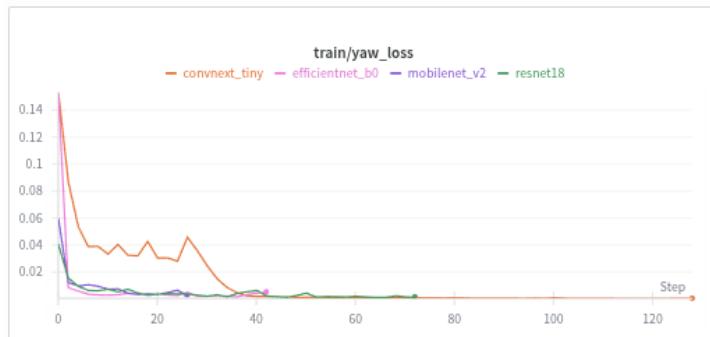


FIGURE 7.47 – Train / Yaw Loss



FIGURE 7.48 – Train / Total Loss

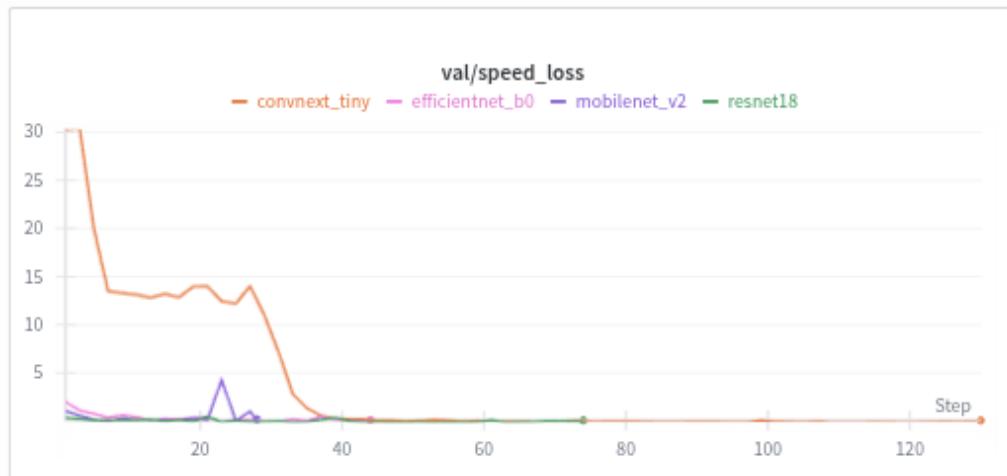


FIGURE 7.49 – Val / Speed Loss

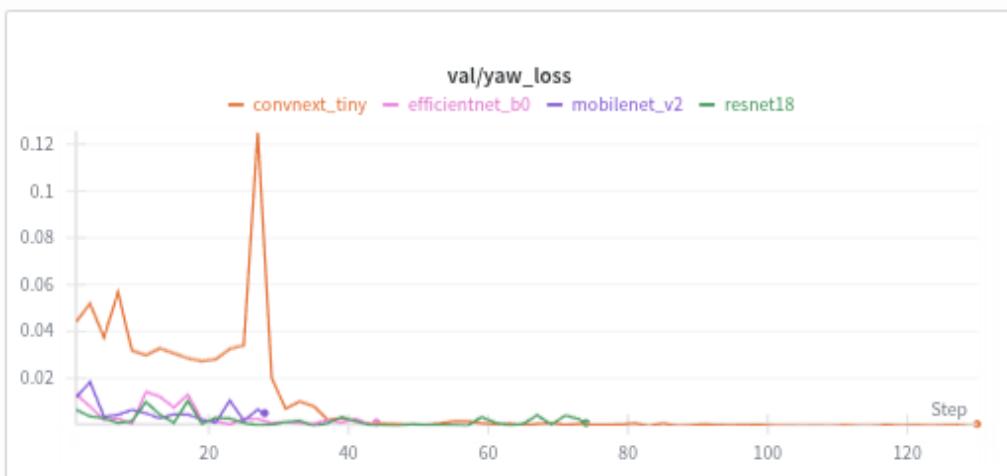


FIGURE 7.50 – Val / Yaw Loss

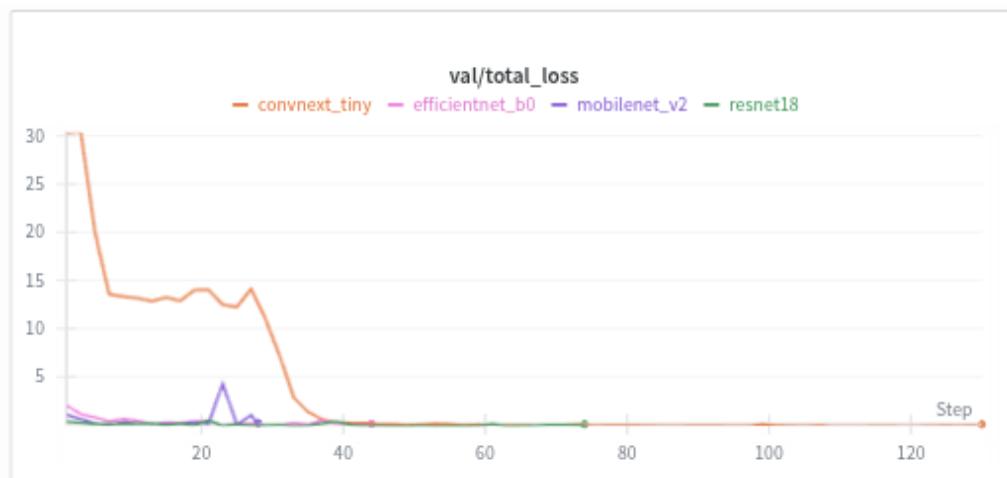


FIGURE 7.51 – Val / Total Loss

7.2 RTMaps

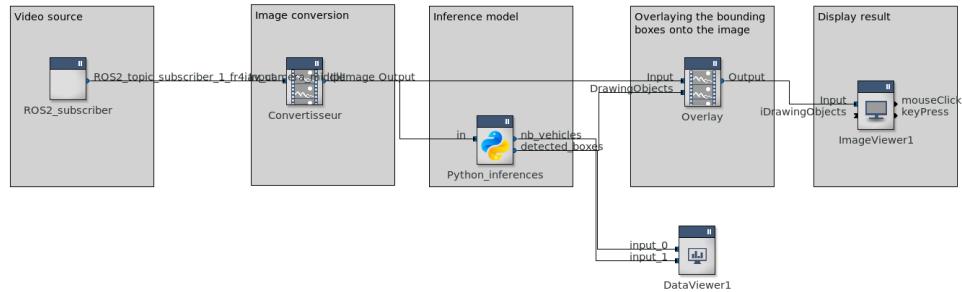


FIGURE 7.52 – Diagramme RTMaps en temps réel

Voici le fonctionnement du `PythonBridge`, qui réalise les inférences sur chaque image de l'entrée du diagramme.

Il est divisé en plusieurs méthodes principales :

- **Dynamic()** : cette fonction est appelée au moment de la déclaration du composant. Elle permet de définir dynamiquement les propriétés du module (choix des modèles YOLO et YawSpeed, seuil de confiance, paramètres de caméra), ainsi que les entrées et sorties du flux de données RTMaps.
- **Birth()** : exécutée une seule fois à l'initialisation du composant, elle charge le modèle YOLO utilisé pour la détection d'objets, ainsi que le modèle de régression Yaw/Vitesse adapté. Elle initialise aussi les fonctions de prétraitement des images (redimensionnement, normalisation), ainsi que les paramètres optiques nécessaires au calcul des distances.
- **Core()** : c'est la boucle principale appelée à chaque nouvelle image en entrée. Elle applique YOLO pour détecter les véhicules, puis estime l'angle de lacet (yaw) et la vitesse grâce au modèle de régression. Les distances approximatives sont calculées à partir des dimensions de la boîte englobante et des paramètres caméra. Enfin, les résultats sont envoyés sous forme de sorties RTMaps : boîtes de détection, flèches représentant l'orientation, et texte affichant les valeurs estimées.
- **Death()** : cette fonction est appelée à la destruction du composant. Elle permet de libérer les ressources associées, même si dans notre implémentation actuelle, elle n'a pas besoin de consignes supplémentaires puisque les ressources sont allouées dynamiquement dans Core()

Ainsi, le composant `rtmaps_python` encapsule toute la logique de détection et d'inférence en exploitant les réseaux YOLO pour la localisation et des modèles légers pour la régression de la vitesse et de l'orientation. Voici d'abord le code de celui-ci :

Listing 7.1 – Extrait du composant RTMaps

```

1  classNames = ["f1car"]
2  MODEL_CLASSES = {
3      "convnext_tiny": YawSpeedDistRegressor_ConvNeXtTiny,
4      "efficientnet_b0": YawSpeedDistRegressor_EfficientNetB0,
5      "mobilenet_v2": YawSpeedDistRegressor_MobileNetV2,
6      "resnet18": YawSpeedDistRegressor_ResNet18,
7      "mobilenet_v3_small": YawSpeedDistRegressor_MobileNetV3Small,
8      "convnext_small": YawSpeedDistRegressor_ConvNeXtSmall,
9      "efficientnet_b1": YawSpeedDistRegressor_EfficientNetB1,
10 }
11
12
13 class rtmmaps_python(BaseComponent):
14     """
15         Composant RTMaps pour la détection de véhicules et l'estimation de
16         la direction (yaw), de la vitesse et de la distance à partir d'
17         une image.
18
19         Propriétés dynamiques :
20             - model: Chemin vers le modèle YOLO (.pt) pour la détection.
21             - yaw_speed_distance_model: Chemin vers le modèle de régression
22                 yaw/vitesse/distancce (.pth).
23             - confidence: Seuil de confiance pour YOLO.
24             - useGPU: Booléen pour utiliser le GPU si disponible.
25             - display_boxes: Affichage des boîtes englobantes.
26             - display_yaw_speed_distance: Affichage de la flèche indiquant
27                 le yaw.
28     """
29
30     def __init__(self):
31         """Initialise le composant RTMaps."""
32         BaseComponent.__init__(self)
33
34     def Dynamic(self):
35         """
36             Définit dynamiquement les propriétés, entrées et sorties du
37             composant.
38         """
39
40         self.add_property("model", "YOLO .pt", rtmmaps.types.FILE)
41         self.add_property("yaw_speed_distance_model", "YawSpeedDistance
42                         .pth", rtmmaps.types.FILE)
43         self.add_property("confidence", 0.85)
44         self.add_property("useGPU", True)
45         self.add_property("display_boxes", True)
46         self.add_property("display_yaw_speed_distance", True)
47
48         self.add_input("in", rtmmaps.types.IPL_IMAGE)
49         self.add_output("nb_vehicles", rtmmaps.types.AUTO)
50         self.add_output("detected_boxes", rtmmaps.types.DRAWING_OBJECT,
51                         buffer_size=50)
52
53     def Birth(self):
54         """
55             Initialise le composant à la naissance.
56
57             - Charge le modèle YOLO pour la détection.
58             - Charge dynamiquement le modèle de régression yaw/vitesse/
59

```

```

        distance selon MODEL_CLASSES.
51 - Initialise les transformations d'image.
52 - Définit les paramètres de la caméra et d'affichage.
53 """
54
55 useGPU = self.properties["useGPU"].data
56 self.myDevice = 'cpu'
57 if useGPU and torch.cuda.is_available():
58     torch.cuda.set_device(0)
59     self.myDevice = 0
60     print("GPU is available and being used")
61 else:
62     print("Using CPU only")
63
64 # YOLO detection model
65 self.weight = self.properties["model"].data
66 self.model = YOLO(self.weight)
67 self.confidence = self.properties["confidence"].data
68 self.colors = [[random.randint(0, 255) for _ in range(3)] for _
69   in classNames]
70
71 # Yaw/Speed model
72 yaw_speed_weights = self.properties["yaw_speed_distance_model"].
73   data
74 # Déterminer le type de modèle depuis le nom du fichier (ex: "resnet18_best.pth")
75 model_name = None
76 for key in MODEL_CLASSES.keys():
77     if key in yaw_speed_weights.lower():
78         model_name = key
79         break
80 if model_name is None:
81     raise ValueError(f"Nom du modèle inconnu dans MODEL_CLASSES pour {yaw_speed_weights}")
82
83 ModelClass = MODEL_CLASSES[model_name]
84 self.yaw_speed_distance_model = ModelClass()
85 self.yaw_speed_distance_model.load_state_dict(torch.load(
86     yaw_speed_weights, map_location=self.myDevice))
87 self.yaw_speed_distance_model.to(self.myDevice)
88 self.yaw_speed_distance_model.eval()
89
90 # Prétraitement images
91 self.preprocess = T.Compose([
92     T.ToPILImage(),
93     T.Resize((224, 224)),
94     T.ToTensor(),
95     T.Normalize(mean=[0.485, 0.456, 0.406],
96                 std=[0.229, 0.224, 0.225]),
97 ])
98
99 self.display_boxes = bool(self.properties["display_boxes"].data)
100 self.display_yaw_speed_distance = bool(self.properties["display_yaw_speed_distance"].data)
101
102 def Core(self):
103 """
104 Boucle principale du composant RTMaps.

```

```

102     - Récupère l'image d'entrée.
103     - Effectue la détection de véhicules avec YOLO.
104     - Pour chaque véhicule détecté :
105         - Estime le yaw, la vitesse et la distance avec le modèle de
106             régression.
107         - Ajoute les informations visuelles dans 'detected_boxes'.
108     - Envoie le nombre de véhicules détectés et les objets dessinés
109         sur les sorties.
110 """
111
112     img = self.inputs["in"].ioelt.data.image_data
113     h, w = img.shape[:2]
114
115     results = self.model.predict(
116         np.array(img),
117         device=self.myDevice,
118         stream=False,
119         conf=self.confidence
120     )[0]
121
122     detected_boxes = rtmapping.types.Ioelt()
123     detected_boxes.data = []
124     nb_vehicles = 0
125     i = 0
126
127     img_tensor = self.preprocess(img).unsqueeze(0).to(self.myDevice)
128
129     for box in results.boxes:
130         cls = int(box.cls[0])
131         if cls == 0:
132             nb_vehicles += 1
133
134         r_c, g_c, b_c = self.colors[cls]
135         x1, y1, x2, y2 = map(int, box.xyxy[0])
136
137         x1, y1 = max(0, x1), max(0, y1)
138         x2, y2 = min(w, x2), min(h, y2)
139
140         box_w = x2 - x1
141         box_h = y2 - y1
142         bbox_norm = torch.tensor([
143             (x1 + box_w / 2) / w,
144             (y1 + box_h / 2) / h,
145             box_w / w,
146             box_h / h
147         ], dtype=torch.float32).unsqueeze(0).to(self.myDevice)
148
149         if self.display_yaw_speed_distance:
150             with torch.no_grad():
151                 sin_yaw, cos_yaw, vx, vy, distance = self.
152                     yaw_speed_distance_model(img_tensor, bbox_norm).
153                     cpu().numpy().flatten()
154
155             yaw = math.atan2(sin_yaw, cos_yaw)
156             distance = distance
157
158             # BBOX
159             if self.display_boxes:
160                 detected_boxes.data.append(rtmapping.types.DrawingObject())
161                 detected_boxes.data[i].kind = 2

```

```
156         detected_boxes.data[i].color = RGBTo32bitInt(r_c, g_c,
157                                         b_c)
158         detected_boxes.data[i].width = 3
159         rect = rtmags.types.Rectangle()
160         rect.x1, rect.y1, rect.x2, rect.y2 = x1, y1, x2, y2
161         detected_boxes.data[i].data = rect
162         i += 1
163
164     # Flèche yaw
165     if self.display_yaw_speed_distance:
166         cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
167         length = 50
168         yaw_display = yaw - math.pi / 2
169         tip_x = int(cx + length * math.cos(yaw_display))
170         tip_y = int(cy + length * math.sin(yaw_display))
171         detected_boxes.data.append(rtmags.types.DrawingObject())
172         detected_boxes.data[i].kind = 1
173         detected_boxes.data[i].color = RGBTo32bitInt(r_c, g_c,
174                                         b_c)
175         detected_boxes.data[i].width = 2
176         line = rtmags.types.Line()
177         line.x1, line.y1, line.x2, line.y2 = cx, cy, tip_x,
178                                         tip_y
179         detected_boxes.data[i].data = line
180         i += 1
181
182     # Texte infos
183     detected_boxes.data.append(rtmags.types.DrawingObject())
184     txt = rtmags.types.Text()
185     txt.text = f"Y:{(yaw*180/math.pi):.2f} VX:{vx:.2f} VY:{vy:.2
186     f} D:{distance:.2f}m" if self.display_yaw_speed_distance
187     else f"None"
188     detected_boxes.data[i].data = txt
189     i += 1
190
191     self.write("detected_boxes", detected_boxes)
192     self.write("nb_vehicles", nb_vehicles)
193
194     def Death(self):
195         """
196             Fonction appelée à la destruction du composant.
197         """
198         pass
```

7.3 Interface Streamlit

7.3.1 Application d'annotation

Afin de rendre plus accessible et visuel le processus d'annotation et de prédiction, une interface interactive a été développée en Streamlit. Cette interface permet de parcourir le dataset d'images, de visualiser directement les résultats des annotations automatiques (boîtes englobantes, labels, etc.), et de comparer en un clic l'image brute et son équivalent annoté.

Fonctionnement général Au préalable, le dataset doit être généré automatiquement, sinon la visualisation ne fonctionne pas. Le fonctionnement de l'application se décompose alors en deux étapes :

1. L'interface Streamlit affiche l'image brute et, au choix, l'image annotée correspondante.
2. Des boutons permettent de naviguer dans les images (\leftarrow , \rightarrow) et d'afficher/masquer les annotations.

Exemple d'affichage La figure 7.53 présente un exemple concret de l'interface :

- En haut : l'image brute issue du dataset.
- En bas : l'image annotée automatiquement avec les boîtes englobantes, ainsi que l'orientation et la distance à titre informatif.



FIGURE 7.53 – Exemple d'affichage Streamlit : comparaison entre image brute et image annotée.

7.3.2 Application d'inférence

Ce module fournit une interface simple et efficace pour :

1. Explorer différents modèles.
2. Tester directement sur des images fournies par l'utilisateur.
3. Visualiser de manière combinée les résultats qualitatifs (images annotées) et quantitatifs (valeurs numériques).

Présentation générale Ce module Streamlit permet à l'utilisateur de :

- Sélectionner un modèle YOLO entraîné pour la détection d'objets sur notre dataset
- Choisir un modèle de régression Yaw/Speed/Distance.
- Fournir en entrée soit une image unique, soit un dossier d'images.
- Visualiser les résultats de l'inférence directement dans l'interface web :
 - Images avec boîtes englobantes.
 - Données numériques (angle, vitesse, distance) correspondantes affichées à côté.

Pipeline de fonctionnement Le schéma général du module est le suivant :

$$\text{Image brute} \xrightarrow{\text{YOLO}} \text{Détection (bounding boxes)} \xrightarrow{\text{Yaw/Speed/Distance}} \text{Angle, Vitesse, Distance}$$

Exemple d'affichage

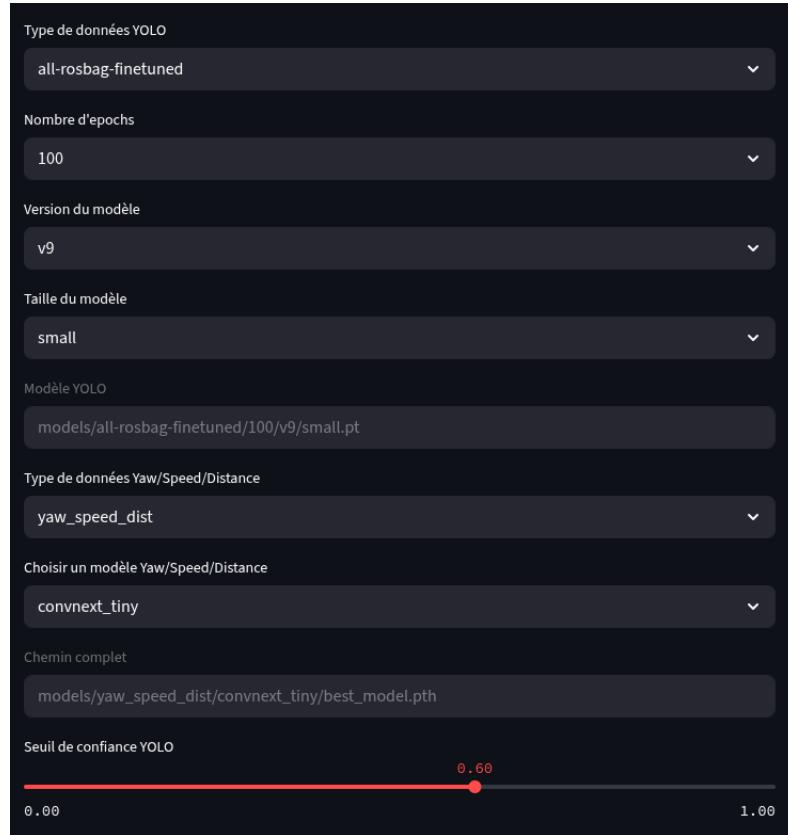


FIGURE 7.54 – Choix des modèles disponibles et seuil de confiance

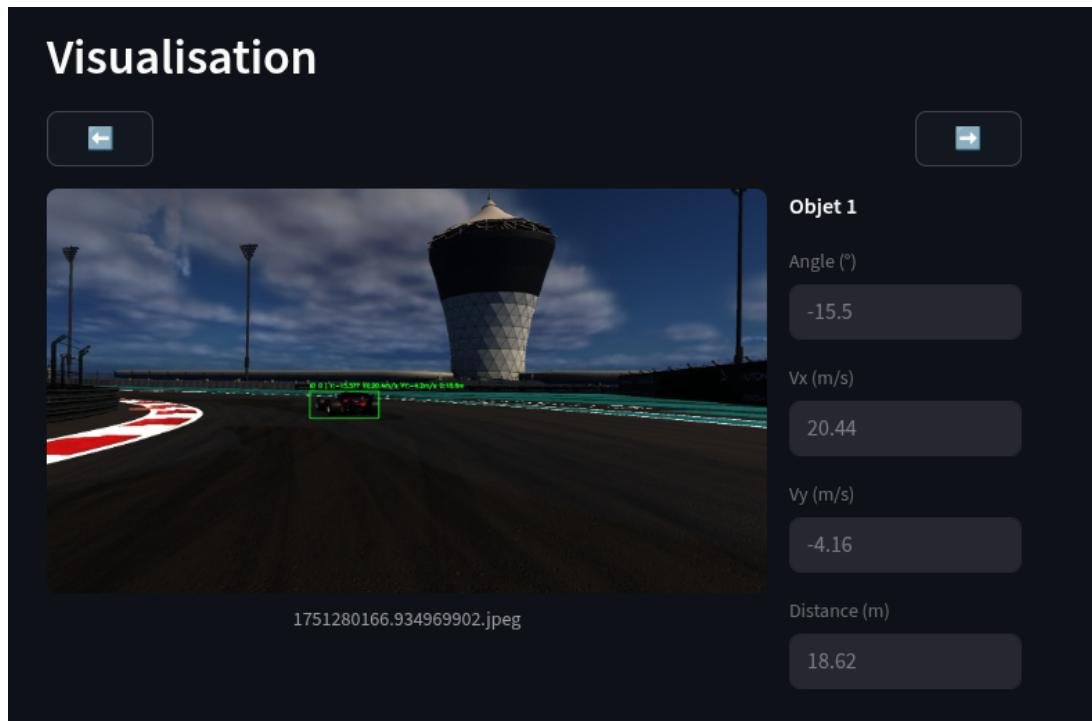


FIGURE 7.55 – Exemple de visualisation avec le module d'inférence.

Chapitre 8

Quatrième de couverture

Sommaire

8.1 Résumé	66
8.2 Abstract	66

8.1 Résumé

Ce rapport présente le développement d'une chaîne de traitement pour la détection et le suivi des voitures adverses dans un contexte de course automobile automatisée. Le projet s'appuie sur des technologies avancées de vision par ordinateur, notamment les modèles de détection classiques basés sur des réseaux de neurones profonds, tels que YOLO et MMDetection. Un modèle personnalisé a été intégré pour estimer l'angle et la vitesse des véhicules détectés, complété par un filtre de Kalman permettant un suivi précis des objets à travers les images successives. Le pipeline développé inclut également l'extraction et la génération automatique de données d'annotations à partir d'un simulateur, ainsi qu'un module RTMaps permettant l'intégration dans le véhicule. Les solutions mises en œuvre offrent un compromis entre rapidité, précision et adaptabilité, adaptées à une application embarquée temps réel. Les résultats montrent une amélioration significative de la robustesse de la détection et du suivi dans des conditions variées.

8.2 Abstract

This report presents the development of a processing pipeline for detecting and tracking opposing cars in an automated racing environment. The project leverages advanced computer vision technologies, notably classical detection models based on deep neural networks such as YOLO and MMDetection. A custom model was incorporated to estimate the angle and speed of detected vehicles, complemented by a Kalman filter for accurate object tracking across successive frames. The pipeline also includes automatic extraction and generation of annotation data from a simulator, as well as an RTMaps module enabling integration into the vehicle. The implemented solutions strike a balance between speed, accuracy, and adaptability, suitable for real-time embedded applications. The results demonstrate a significant improvement in detection and tracking robustness under varied conditions.