Rapport de Stage d'Application

Stage 2A

Amélioration de la détection de collisions entre usagers vulnérables et véhicules

Étudiant stagiaire :
Mohamed Dyae CHELLAF

Tuteur de stage :
Sabri KHAMARI

Rapporteur de stage :
Julien ALLALI

Entreprise :
ENSEIRB-MATMECA

Effectué à l'ENSEIRB-MATMECA
16 juin 2025 - 25 juillet 2025 & 18 août 2025 - 26 septembre 2025

Remerciements

Je tiens à exprimer ma profonde gratitude envers toutes les personnes qui m'ont apporté leur soutien et leurs précieux conseils tout au long de ce stage :

— Je tiens à adresser mes sincères remerciements à Monsieur Mohamed MOSBAH, qui m'a proposé ce stage et m'a offert l'opportunité d'intégrer l'équipe des chercheurs de la chaire Mobilité et Transports Intelligents. Sa confiance et son soutien ont constitué un véritable moteur pour la réalisation de ce travail, et m'ont permis de progresser tant sur le plan académique que professionnel.

— Je souhaite exprimer toute ma gratitude à Monsieur Sabri KHAMARI, mon tuteur de stage, pour son accompagnement constant et ses conseils techniques précieux. Grâce à son encadrement rigoureux et à ses explications, j'ai pu approfondir mes connaissances sur les systèmes de transport intelligents coopératifs (`C-ITS`) et acquérir une expérience enrichissante tout au long de ce stage.

— Je tiens également à remercier Monsieur Julien ALLALI, rapporteur de mon stage, pour son suivi attentif et ses retours toujours constructifs sur l'ensemble des documents que j'ai fournis. Ses remarques pertinentes et ses conseils avisés m'ont permis d'améliorer la qualité de mes travaux et d'enrichir considérablement ce rapport.

Résumé

En 2025, j'ai intégré l'équipe de recherche de la chaire **Mobilité et Transports Intelligents de Bordeaux**, dans le cadre d'un stage consacré à la détection de collisions entre usagers vulnérables (cyclistes, piétons) et véhicules motorisés à partir des messages `C-ITS` **(Cooperative Intelligent Transport Systems)**. L'objectif de ce stage était de concevoir et développer un système permettant d'identifier de manière fiable et rapide les situations à risque en milieu urbain, où la sécurité routière constitue un enjeu majeur.

Le sujet portait sur la mise en place d'un **algorithme de détection de collisions** exploitant les messages `CAM` **(Cooperative Awareness Message)**. Le projet comportait plusieurs volets : prétraitement des données (correction des erreurs GPS, gestion de la latence et des valeurs aberrantes), conception et comparaison de méthodes heuristiques et de modèles de *machine learning*, ainsi que l'évaluation de leur efficacité sur des jeux de données expérimentaux.

La solution développée s'appuie sur une architecture modulaire en `Python`, intégrant un pipeline de traitement des données (nettoyage, extraction de paires de proximité, génération de labels, reconstruction de trajectoires) et un module d'apprentissage supervisé permettant d'entraîner et d'évaluer différents modèles.

En parallèle, une **interface WEB interactive** a été réalisée à l'aide de `React, Next.js` **et** `Leaflet` afin de visualiser les alertes et les prédictions de l'algorithme sur une carte. Cette interface offre notamment la possibilité de filtrer les alertes par `station_id` et par type d'alerte, et de comparer en temps réel les résultats obtenus sur différents scénarios.

Ce projet m'a permis de renforcer mes compétences en **traitement de données, algorithmique et *machine learning***, tout en découvrant les enjeux techniques et scientifiques liés aux `C-ITS` et à la sécurité des usagers vulnérables.

Table des matières

1 Introduction

1.1 Présentation de l'ENSEIRB-MATMECA et du LaBRI.

ENSEIRB-MATMECA, école d'ingénieurs de Bordeaux INP située à Talence, est une institution de formation et de recherche de référence en France. Née de la fusion en 2009 de l'ENSEIRB (Électronique, Informatique, Télécommunications) et de MATMECA (Mathématiques et Mécanique), elle propose aujourd'hui des formations d'excellence dans des domaines variés tels que l'électronique, l'informatique, les télécommunications, la mécanique et le calcul scientifique. L'école se distingue par son lien étroit avec la recherche et l'industrie, favorisant l'innovation et la professionnalisation de ses étudiants.

Le **LaBRI** (Laboratoire Bordelais de Recherche en Informatique), unité mixte de recherche associée à l'Université de Bordeaux, au CNRS et à Bordeaux INP, constitue un acteur majeur de la recherche en informatique en Nouvelle-Aquitaine. Ses thématiques couvrent un large spectre, allant des fondements théoriques de l'informatique jusqu'aux applications pratiques, notamment en algorithmique, intelligence artificielle, systèmes distribués, traitement de données massives et interaction homme-machine. Le LaBRI participe activement à des collaborations nationales et internationales, renforçant ainsi le rayonnement scientifique et académique de la région bordelaise.

1.2 Contexte du stage et de la chaire Mobilité et Transports Intelligents

Le stage s'inscrit dans le cadre des travaux menés au sein de la **Chaire Mobilité et Transports Intelligents** de Bordeaux, un programme de recherche et d'innovation dédié à l'étude et au développement de solutions pour améliorer la mobilité urbaine et la sécurité routière. Cette chaire, portée par un partenariat entre des acteurs académiques et industriels, a pour vocation de concevoir des technologies permettant de rendre les déplacements plus sûrs, fluides et respectueux de l'environnement. Dans ce contexte, la détection de situations à risque impliquant les usagers vulnérables, tels que les cyclistes et les piétons, constitue un enjeu central.

2 Besoin et mission technique

2.1 Enjeux de la sécurité routière et du C-ITS

La sécurité routière est un enjeu majeur dans les politiques de mobilité urbaine, en particulier face à la vulnérabilité croissante de certains usagers comme les cyclistes et les piétons. L'augmentation des modes de transport doux et partagés, combinée à la densité du trafic urbain, accentue les risques d'accidents dans les environnements complexes et dynamiques. Dans ce contexte, les **Systèmes de Transport Intelligents Coopératifs** (`C-ITS`) apparaissent comme une solution prometteuse pour améliorer la prévention des collisions.

Les `C-ITS` permettent aux véhicules, aux infrastructures et aux usagers d'échanger des informations en temps réel grâce à des messages standardisés tels que les `CAM` (*Cooperative Awareness Messages*) ou les `DENM` (*Decentralized Environmental Notification Messages*). Ces messages visent à accroître la perception de l'environnement pour les conducteurs et les systèmes embarqués, en fournissant des données précises sur la position, la vitesse ou les événements critiques à proximité. En rendant les interactions plus transparentes et anticipées, les `C-ITS` renforcent la réactivité du système de transport dans son ensemble et contribuent ainsi à réduire les situations dangereuses et les accidents.

2.2 Présentation des messages `CAM` et `DENM`

Les messages `CAM` et `DENM` sont deux types de messages normalisés dans le cadre des systèmes `C-ITS`.

Le message `CAM` est un message périodique émis par les véhicules pour partager leur état courant avec les autres acteurs à proximité. Il contient notamment des informations telles que la position GPS, la vitesse, le cap, l'identifiant de station, ainsi que des attributs dynamiques du véhicule. Ce message permet d'assurer une conscience coopérative de l'environnement et constitue une source essentielle pour la détection de proximité entre véhicules ou avec des usagers vulnérables.

Le message `DENM`, quant à lui, est émis de manière événementielle, lorsqu'un danger ou une situation anormale est détectée (freinage brusque, obstacle, accident, conditions météo critiques, etc.). Il vise à alerter les autres usagers de la route afin qu'ils puissent adapter leur comportement.

2.3 Le besoin du projet

Ce projet est né du besoin de mieux exploiter les messages d'alerte collectés lors des expérimentations de mobilité urbaine, disponibles sous forme de fichiers `JSON`. Ces alertes, issues des messages `CAM` et `DENM`, contiennent des informations critiques sur les situations à risque impliquant des usagers vulnérables comme les cyclistes. Afin de les

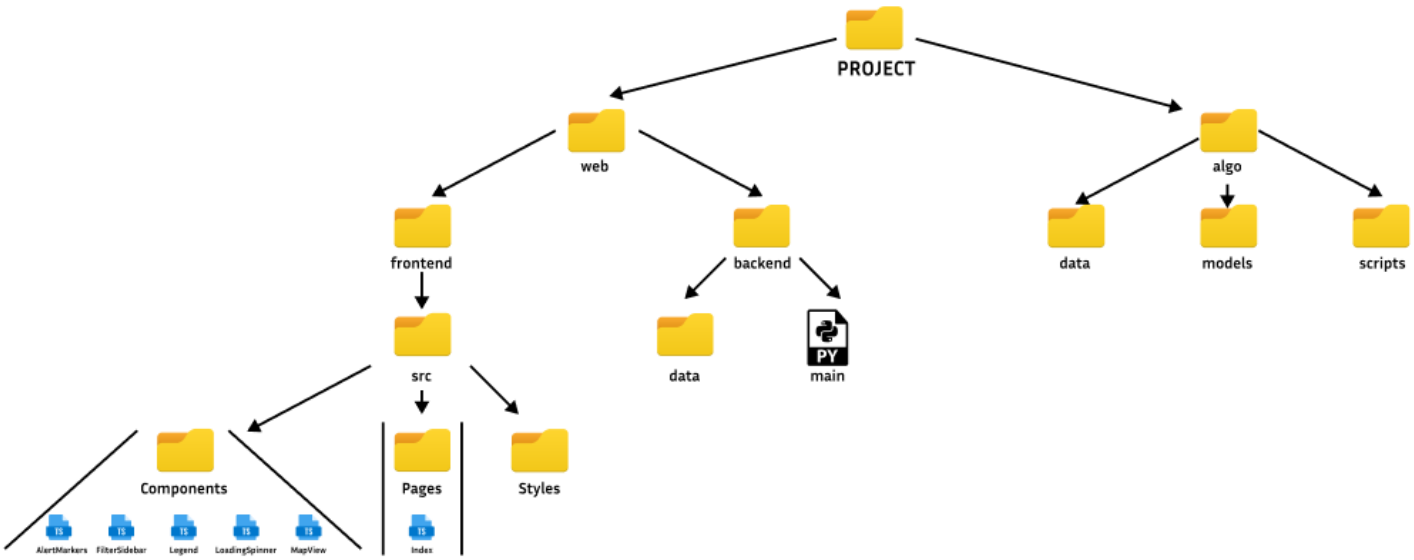
analyser efficacement, deux outils complémentaires ont été nécessaires : d'une part, **outil de visualisation** tel qu'**une interface web** intuitive pour les visualiser dynamiquement sur carte ; d'autre part, **un algorithme** pour prédire automatiquement les risques de collision à partir de ces données. Le projet vise ainsi à transformer ces données brutes en informations exploitables pour la recherche et la sécurité routière.

2.4 Mission confiée dans le cadre du projet

Dans le cadre de ce stage, plusieurs missions m'ont été confiées, avec pour objectif de contribuer à l'analyse, au traitement et à l'exploitation des messages `C-ITS`, en vue de détecter automatiquement des situations de collision. La première étape a consisté à développer une **interface web interactive** permettant de visualiser les messages `CAM` sur une carte, avec différentes options de filtrage et de représentation. Cette interface a servi de support essentiel pour mieux appréhender les données collectées et faciliter leur inspection par les encadrants. Par la suite, une seconde mission a porté sur la conception d'un **pipeline algorithmique de détection**, reposant sur le prétraitement des données (correction GPS, suppression d'anomalies dans les données), la génération automatique de paires de proximité, la création de jeux de données labellisés, et l'implémentation de méthodes heuristiques ainsi que de modèles de machine learning pour prédire les situations de risque. Ces deux volets, bien que développés séparément, ont été conçus dans une logique complémentaire : visualisation et interprétation des données d'un côté, traitement algorithmique et détection d'événements critiques de l'autre.

3 Spécification et conception du système

3.1 Structure et arborescence du projet



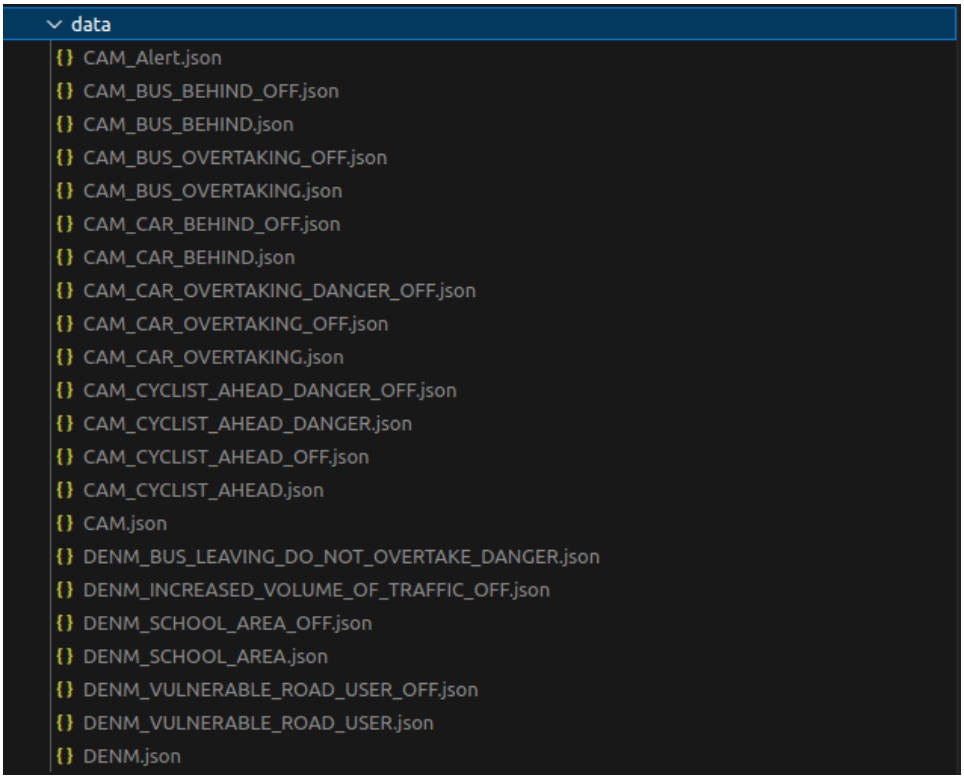
3.2 Prétraitement des données

Le prétraitement des données a débuté par l'écriture d'un *script* en `Python` dédié à l'analyse du fichier brut `CAM.json`, contenant l'ensemble des messages `CAM` collectés durant les expérimentations effectuées avec les usagers. L'objectif de cette première étape était d'extraire uniquement les messages contenant des alertes spécifiques, identifiables par la présence des champs `eyenet_alert` ou `denm_alert` dans le corps du message.

Le *script* a parcouru l'ensemble des messages *JSON* du fichier `CAM.json`, et a généré deux fichiers filtrés, un premier nommé `CAM_Alert.json` pour les `eyenet_alert`, et un deuxième nommé `DENM_Alert.json` pour les `denm_alert`. Chaque message valide présente généralement la structure suivante :

```
{
  "timestamp": "2025-06-18T14:52:01.123Z",
  "station_id": 1001,
  "position": {
    "latitude": 44.808746,
    "longitude": -0.603866
  },
  "speed": 12.5,
  "heading": 75.0,
  "eyenet_alert": "CAR_BEHIND",
  "denm_alert": "SCHOOL_AREA" // Peut ne pas être présent dans certains messages
}
```

Dans un second temps, un traitement complémentaire a été effectué pour **séparer automatiquement les messages par type d'alerte**, afin de générer un fichier `.json` par alerte. Les fichiers générés à partir du traitement ont été organisés dans le répertoire `web/backend/data` comme suit :



Types d'alerte

Un nettoyage complémentaire a ensuite été appliqué pour retirer les messages incomplets : absence de coordonnées *GPS*, vitesses nulles ou incohérentes, ou *timestamp* non chronologiques. Ce filtrage a permis d'assurer la qualité et la fiabilité des données utilisées dans le reste du projet.

3.3 Conception de l'interface graphique

Cette phase de conception a commencé par l'identification des besoins fonctionnels : affichage des messages géolocalisés, différenciation par type d'alerte, filtrage par identifiant de station, et affichage de métadonnées utiles (*timestamp*, vitesse, direction, etc.).

Des maquettes ont ensuite été réalisées à l'aide de la plateforme *Figma*, afin de visualiser l'agencement des composants (carte, *sidebar*, légende, filtres) et d'anticiper les interactions utilisateur. Ces prototypes ont permis de valider l'architecture générale de l'interface avant le début du développement, tout en garantissant une cohérence entre les objectifs techniques du projet et l'expérience utilisateur attendue.

3.4 Conception du Back-End

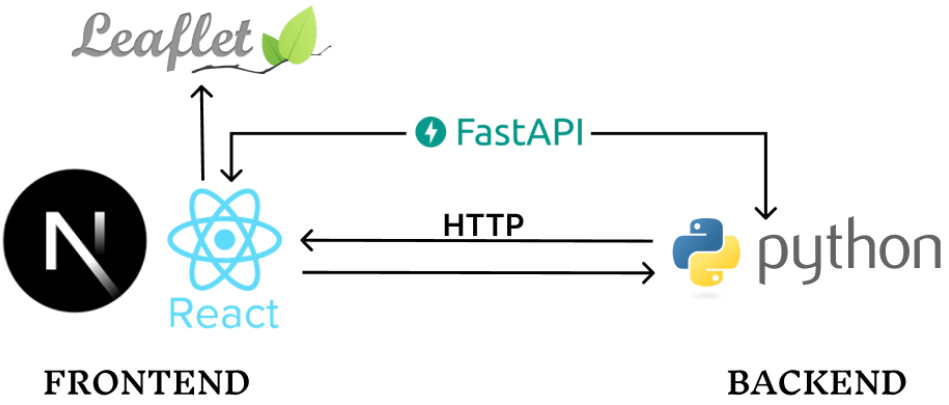
Le backend du projet a été conçu comme un serveur léger en `Python`, destiné à fournir les fichiers de données nécessaires à l'interface de visualisation. Son rôle principal est de servir dynamiquement les messages `CAM` filtrés par type d'alerte, organisés en fichiers `.json`, depuis le répertoire `web/backend/data`.

3.5 Stack technique

Le projet repose sur trois composants principaux : une interface en `Next.js` et `React` pour la visualisation des messages `CAM` et `DENM`, un *backend* en `FastAPI` exécuté avec `Uvicorn` pour servir les fichiers `.json`, et une suite de scripts en `Python` dédiée au prétraitement et aux analyses algorithmiques. Cette organisation assure une séparation claire entre visualisation, service de données et logique métier.

Le choix de `Next.js` et `React` s'inscrit dans la continuité des outils utilisés au sein du projet PFA, et permet de bénéficier d'une structure modulaire, d'un rendu rapide côté client, ainsi que d'une prise en main efficace pour l'affichage dynamique de données. `FastAPI`, quant à lui, a été retenu pour sa légèreté et sa capacité à servir rapidement des fichiers statiques via des routes `HTTP` bien définies.

Aucune base de données n'a été intégrée dans le projet, car la finalité de la partie web est principalement exploratoire : il s'agit d'offrir une interface simple pour naviguer dans les fichiers de messages déjà filtrés, sans besoin de persistance ou de traitement transactionnel. Le système repose ainsi sur des fichiers `.json` générés à l'avance, chargés à la demande selon les filtres appliqués par l'utilisateur.



Stack technique de l'interface graphique

4 Réalisation et développement

4.1 Développement de l'interface graphique (`/web`)

4.1.1 Front-End

L'interface graphique se présente sous la forme d'une carte interactive accompagnée d'une barre latérale permettant de filtrer dynamiquement les alertes. L'utilisateur peut visualiser :

- toutes les alertes émises par une station donnée (`station_id`) ;
- un type d'alerte spécifique associé à une station ;
- un type d'alerte, indépendamment de la station.

Les types d'alertes sont représentés par des couleurs distinctes, et une légende s'affiche automatiquement lorsque plusieurs types sont visibles simultanément sur la carte.

Les alertes sont représentées par des cercles géolocalisés (***markers***) : un clic sur un marqueur affiche un encart contenant les détails de l'alerte (station, type, coordonnées, timestamp, etc.). Ce fonctionnement vise à offrir une navigation fluide et intuitive au sein des données collectées.

L'interface a été développée en suivant une approche basée sur des composants réutilisables et une gestion locale de l'état. Le composant principal `MapView.tsx` agit comme point central de l'affichage cartographique et orchestre les autres sous-composants.

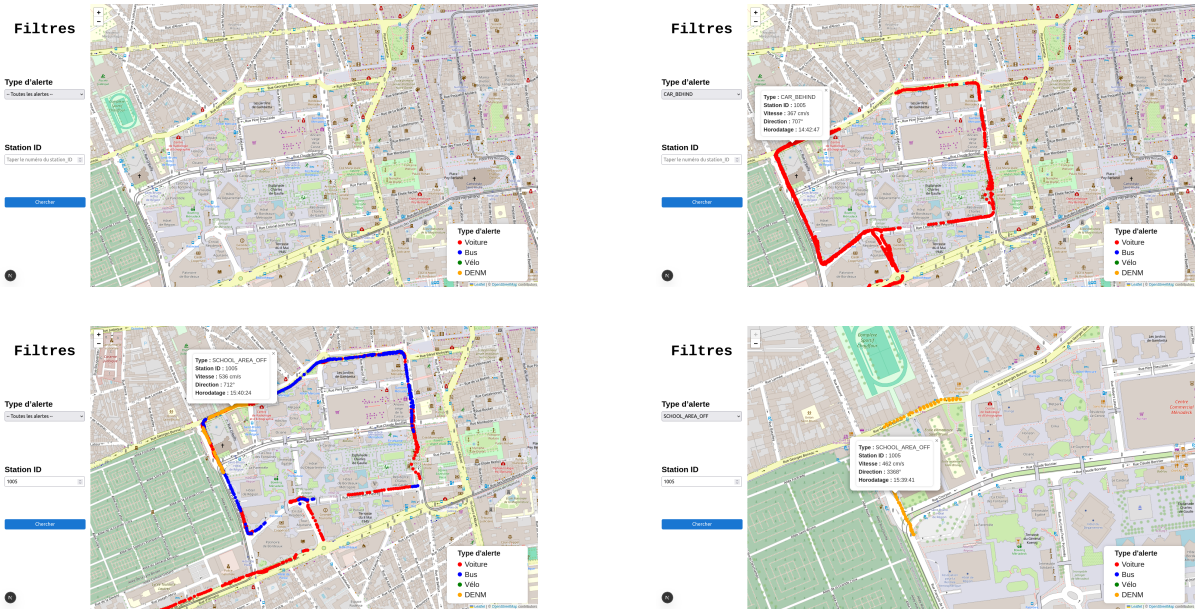
Lors du chargement, `MapView.tsx` déclenche des requêtes `HTTP` vers le backend *FastAPI* à l'aide du *hook* `useEffect`, afin de récupérer dynamiquement les fichiers `.json` associés aux types d'alertes. Ces données sont ensuite stockées localement via le `useState` de React.

Les messages reçus sont transmis en tant que `props` aux composants enfants, notamment `AlertMarkers.tsx`, qui se charge de l'affichage des cercles de couleur sur la carte, en fonction du type d'alerte. Le filtrage par type d'alerte et par `station_id` est géré via le composant `FilterSidebar.tsx`, qui met à jour l'état global et redirige les données filtrées vers les composants d'affichage.

L'ensemble du flux repose sur une gestion réactive des données côté client, sans rafraîchissement de page, assurant une interface fluide, lisible et synchronisée avec les données disponibles sur le serveur.

Le frontend est exécuté localement à l'aide de `Next.js`. Après installation des dépendances via la commande `npm install`, l'interface est lancée avec `npm run dev`, ce qui démarre le serveur de développement sur le port `3000`. L'application devient alors accessible à l'adresse `http://localhost:3000`.

```
# Lancement de l'interface
npm run dev
```



Quelques captures de l'inteface graphique

4.1.2 Back-End

Le *backend* a été isolé dans un environnement virtuel `venv`, permettant de gérer proprement les dépendances `Python` sans interférer avec le système global. L'installation des modules nécessaires (`fastapi`, `uvicorn`, `pydantic`, etc.) est gérée via `pip` à l'aide d'un fichier `requirements.txt`.

Une fois l'environnement activé, le serveur est lancé localement à l'aide de `uvicorn`, qui joue le rôle de serveur `ASGI` et permet une exécution rapide et non bloquante de l'application `FastAPI`.

```
# Activation de l'environnement virtuel
source venv/bin/activate

# Lancement du serveur FastAPI
uvicorn main:app --host 127.0.0.1 --port 8000 --reload
```

Le backend a été conçu comme une interface de service minimaliste, dont le rôle principal est de fournir à l'interface web les fichiers `.json` contenant les messages `CAM` filtrés. Il a été implémenté avec le framework *FastAPI* et déployé via le serveur *Uvicorn*, pour bénéficier de performances légères et d'une compatibilité native avec les requêtes asynchrones.

Les données sont stockées dans le répertoire `web/backend/data`, où chaque fichier

correspond à un type d'alerte spécifique (par exemple `CAM_CAR_BEHIND.json`). Lorsqu'un utilisateur interagit avec l'interface (filtre par type d'alerte ou `station_id`), une requête `HTTP GET` est envoyée au backend.

Le backend expose plusieurs routes :

- `/alerts` : retourne la liste de tous les fichiers d'alertes disponibles ;
- `/alerts/{filename}` : retourne le contenu d'un fichier `.json` donné ;
- `/alerts/{filename}/station/{station_id}` : retourne uniquement les messages correspondant à un type d'alerte et une station spécifique.

Grâce à la configuration du `CORS`, ces routes peuvent être appelées librement depuis le frontend local. Le backend ne contient aucune logique algorithmique de traitement : il agit uniquement comme un serveur de données, offrant un accès structuré, rapide et filtrable aux messages nécessaires à l'affichage.

4.2 Développement de la partie (`/algo`)

L'algorithme de détection de collisions a été développé sous forme d'un pipeline `Python` entièrement modulaire, où chaque étape produit un fichier `.csv` exploitable par les suivantes. Ce fonctionnement structuré permet de suivre pas à pas la transformation des données brutes en résultats exploitables.

Le choix de cette organisation en étapes successives plutôt qu'un code unique regroupant l'ensemble du traitement, a été motivé par la volonté de rendre le système plus lisible, maintenable et testable. Chaque étape étant indépendante, il devient plus facile d'isoler un problème, de diagnostiquer une erreur ou de modifier une partie du processus sans impacter les autres. Cette séparation permet également de rejouer sélectivement certaines étapes du pipeline (comme l'entraînement ou la prédiction), sans avoir à recharger ou recalculer l'ensemble du flux.

4.2.1 Reconstruction des trajectoires

Le pipeline débute par la reconstitution des trajectoires à partir du fichier `cam_cleaned.csv`, qui contient les messages `CAM` nettoyés. Le script `reconstruct_trajectories.py` trie les données par `station_id` et `timestamp`, et enregistre le fichier `trajectories.csv`. Ce fichier constitue la base temporelle et spatiale du traitement.

4.2.2 Génération des paires de proximité

À partir des trajectoires, le script `find_proximity_pairs.py` identifie toutes les paires d'entités se trouvant à moins de 15 mètres l'une de l'autre dans une fenêtre temporelle de ± 1 seconde. Pour chaque paire, des attributs sont extraits : distance, vitesses respectives, angles de déplacement et coordonnées GPS. Le résultat est enregistré dans `proximity_pairs.csv`.

La distance entre deux entités est calculée via la formule de `Haversine`, prenant en compte la courbure de la Terre :

Listing 1 – Calcul de distance entre deux points GPS

```
def haversine(lat1, lon1, lat2, lon2):
    R = 6371000  # rayon de la Terre en metres
    ...
    return R * c
```

4.2.3 Labellisation automatique des collisions

Le fichier `proximity_pairs.csv` est ensuite enrichi avec un label `collision` à l'aide du script `generate_labels.py`. Cette labellisation repose sur plusieurs règles métier :

- distance < 5 m
- vitesses > 2 m/s
- différence d'angle $< 45°$
- distance latérale < 1.5 m
- position longitudinale entre -5 m et +5 m

Les distances latérales et longitudinales sont calculées en projetant la position relative de l'autre entité par rapport à l'orientation de déplacement (*heading*) du véhicule principal.

Le résultat est un fichier `proximity_pairs_labeled.csv`, contenant une colonne binaire nommée `collision`.

4.2.4 Fusion avec d'autres jeux de données

Pour augmenter la taille et la diversité du jeu d'apprentissage, le script `merge_training_data.py` fusionne les données labellisées avec deux autres fichiers : `piste_cyclable_exemples.csv` et `collisions_evidentes.csv`, dans le but d'ajouter de nouvelles situations.
Seules les colonnes communes sont conservées, et l'ensemble est sauvegardé dans `proximity_pairs_labeled.csv` (mis à jour).

4.2.5 Entraînement du modèle de prédiction

Le fichier fusionné est utilisé pour entraîner un modèle de classification avec `scikit-learn`. Le script `train_model.py` extrait les variables explicatives suivantes : distance, vitesses, directions et positions GPS des deux entités. Les données sont divisées en ensemble d'apprentissage et de test (20% de test, stratifié), puis un modèle `Random Forest` est entraîné et évalué :

Listing 2 – Entraînement du modèle Random Forest

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Un rapport de classification (précision, rappel, F1-score) et une matrice de confusion sont affichés pour évaluer les performances. Le modèle entraîné est ensuite sauvegardé au format `.pkl` dans le fichier `models/collision_model.pkl`.

4.2.6 Prédiction sur de nouvelles données

Pour tester le modèle sur de nouveaux cas, le script `predict_collision.py` charge un fichier `.csv` de paires générées (`pairs/my_pairs.csv`) et applique le modèle sauvegardé. Le résultat est un nouveau fichier `my_collisions.csv` contenant une colonne `predicted_collision`, indiquant les situations jugées critiques par le modèle :

Listing 3 – Application du modèle sur de nouvelles paires

```
model = joblib.load(MODEL_PATH)
df["predicted_collision"] = model.predict(df[features])
```

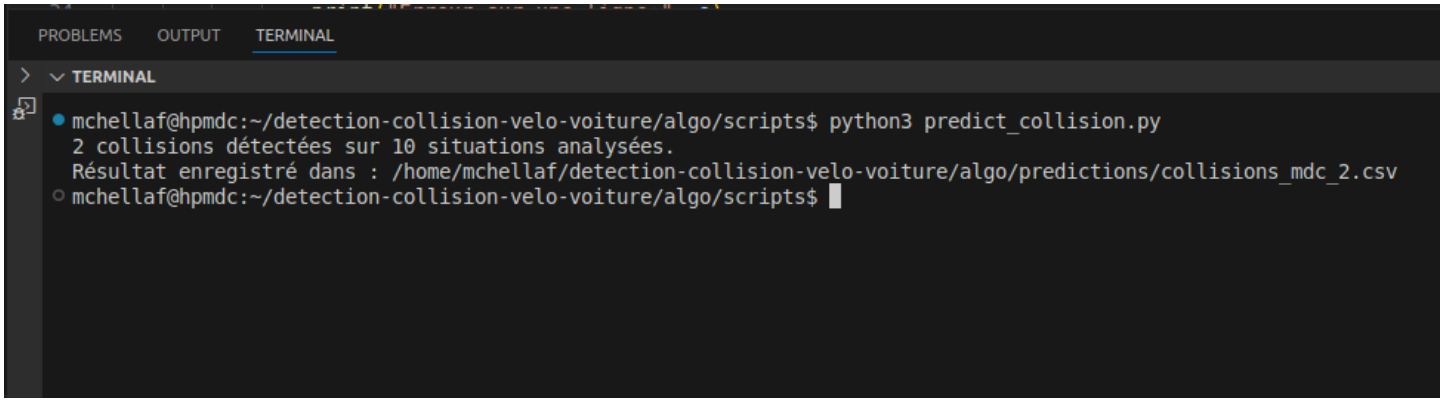
Chaque étape du pipeline repose sur des fichiers intermédiaires `.csv`, ce qui facilite la reproductibilité, la traçabilité et la réutilisation des résultats à chaque phase.

5 Validation et tests

Initialement, des tests devaient être menés à l'aide de l'environnement de simulation `SUMO/Artery`, afin de générer des scénarios synthétiques de circulation et d'évaluer l'algorithme sur des trajectoires simulées avec contrôle précis des interactions. Toutefois, cette piste a été abandonnée dans les deux dernières semaines du stage en raison de contraintes techniques et de délais d'intégration.

En remplacement, une approche de test manuel a été mise en place à l'aide de fichiers `.csv` construits manuellement. Ces jeux de données contiennent des configurations spécifiques (cycliste doublé(e), véhicule en angle, etc.) pour lesquelles le résultat attendu est connu à l'avance. L'algorithme de détection a ensuite été appliqué sur ces cas tests afin de vérifier la robustesse et la cohérence du pipeline.

Ce protocole de test a permis de valider les règles de labellisation, les distances seuils, et la pertinence des prédictions sur des cas concrets, même sans simulation complète.



Sortie de l'algorithme sous format `CSV`

6 Conclusion

6.1 Adéquation de la solution au problème initial

Le projet avait pour objectif principal de concevoir un système complet d'analyse de situations à risque impliquant des usagers vulnérables, à travers deux volets complémentaires : d'une part, une **interface graphique interactive** permettant de visualiser les messages `CAM` collectés sur une carte ; d'autre part, un **outil algorithmique de prédiction** capable de détecter automatiquement les configurations potentiellement dangereuses entre cyclistes et véhicules motorisés.

Le projet a atteint un degré d'achèvement très satisfaisant. L'interface, développée en `Next.js` et `Leaflet`, permet une navigation fluide dans les données, avec des filtres par type d'alerte ou station, et l'affichage des détails de chaque message. De son côté, le pipeline algorithmique en `Python` permet de reconstruire les trajectoires, extraire des paires proches, les labelliser selon des règles définies, entraîner un modèle supervisé, puis l'appliquer sur de nouvelles données.

Des tests manuels ont été menés à partir de fichiers `CSV` construits pour représenter des situations contrôlées, avec des résultats conformes aux attentes. L'ensemble de la solution est donc fonctionnelle, cohérente et en adéquation avec la problématique initiale.

6.2 Limites actuelles et perspectives d'amélioration

Deux principales limites subsistent dans la version actuelle du projet. Tout d'abord, le modèle de prédiction n'a pas été intégré au *backend*, ce qui empêche une utilisation en temps réel dans l'interface. Ensuite, les données issues du fichier `CAM` ne suffisent pas, à elles seules, à représenter l'ensemble des cas réels rencontrés sur le terrain.

Pour la suite, il serait pertinent :

- d'intégrer le modèle dans l'architecture backend pour permettre des prédictions à la volée ;
- d'enrichir les données avec d'autres sources ou capteurs pour améliorer la couverture des cas concrets.

Ces améliorations ouvriraient la voie à une exploitation plus large du système, en lien avec les enjeux de mobilité urbaine et de sécurité routière.

7 Bilan personnel

Ce stage a été une expérience particulièrement formatrice, tant sur le plan technique que méthodologique. Il m'a permis de consolider mes compétences en développement web, notamment avec la bibliothèque `React` et l'écosystème `Next.js`, en travaillant sur une interface interactive complète.

Par ailleurs, la mise en place du pipeline algorithmique m'a permis d'acquérir des connaissances concrètes en *Machine Learning*, en particulier sur les étapes de préparation des données, de labellisation, d'entraînement et d'évaluation de modèles supervisés.

Enfin, ce stage m'a offert une première immersion dans un projet de recherche appliquée au domaine de la mobilité intelligente, et m'a sensibilisé à l'importance des données, de leur qualité, et de leur visualisation dans la compréhension des phénomènes complexes.

8 Annexe : Gestion de projet

Dans le cadre du projet de détection de collisions entre usagers vulnérables et véhicules, une phase initiale de cadrage a été mise en place dès le début du stage. Cette étape, menée en collaboration entre le tuteur et le laboratoire, a permis de définir les grandes lignes du projet ainsi que les objectifs à atteindre. Plusieurs éléments clés ont été clarifiés à cette occasion : le périmètre fonctionnel, la nature des données à exploiter (messages `CAM` et `DENM`), les contraintes techniques, et les attentes de la chaire Mobilité et Transports Intelligents.

Le choix de la stack technologique a également été décidé à ce stade, en s'appuyant sur les outils déjà utilisés dans le cadre du PFA : `Next.js` pour le *frontend*, `FastAPI` pour le *backend*, et `Python` pour la partie algorithmique. Une fois ces choix validés, un découpage du projet en phases successives a été établi : conception de l'interface de visualisation, structuration du *backend*, développement du pipeline de traitement et d'apprentissage, puis phase de test.

Le développement a ensuite progressé de manière itérative, en suivant une méthode de travail inspirée des principes agiles. Des réunions hebdomadaires ont été organisées avec le tuteur de stage, permettant de faire un point sur l'avancement, de valider les modules développés et de réorienter les priorités si nécessaire. Ces échanges réguliers ont favorisé une dynamique de travail souple et efficace, tout en maintenant un alignement fort avec les attentes du laboratoire.

Chaque *sprint* hebdomadaire était consacré à une tâche technique précise (ex. : mise en place de la carte, génération des paires, labellisation, entraînement du modèle, etc.), ce qui a permis d'avancer de manière progressive tout en assurant une cohérence d'ensemble. Cette organisation a été essentielle pour concilier autonomie dans le développement et supervision rigoureuse, dans un projet mêlant traitement de données, développement logiciel et enjeux de mobilité urbaine.