Actifilm

Projet 15 - Activation parallèle de processus cognitifs et affectifs multidimensionnels lors de stimulation écologique par des films

Université de Genève

Cours :
Applications Informatiques

Membres du Groupe :
Viona Cufo
Yardel Hurtado

Juin 2024

Contents

1 Identification des besoins

1.1 Client et utilisateurs

1.1.1 Client

Les clients sont deux chercheurs en neurosciences, Patrik Vuilleumier et Timothée Proix. Leur recherche porte sur l'étude de la représentation cérébrale des émotions. Leur objectif est d'identifier les zones du cerveau correspondant à des émotions spécifiques.

1.1.2 Utilisateurs

Les utilisateurs seront principalement les chercheurs eux-mêmes. Éventuellement, d'autres membres de leur équipe de recherche pourraient utiliser le logiciel pour contribuer à l'analyse des données et à l'interprétation des résultats.

1.2 Problèmes et objectifs

1.2.1 Problèmes rencontrés

L'expérience mise en place par les chercheurs consiste à placer un groupe de personnes individuellement dans un IRMf (imagerie par résonance magnétique fonctionnelle) tout en leur montrant des fragments filmiques. Une fois les données d'activité cérébrale recueillies, la difficulté réside dans la comparaison précise et l'interprétation que les chercheurs doivent effectuer pour analyser leurs résultats.

La difficulté de cette comparaison provient du fait que l'analyse des fragments de films en fonction des différentes caractéristiques qu'ils représentent à chaque moment est une tâche complexe. Les films en tant que stimuli cérébraux, offrent une énorme quantité d'informations difficile à étudier en la comparant aux données de l'activité cérébrale des participants. Ce processus, dans les études classiques sur les émotions, était auparavant réalisé manuellement par des humains, attribuant vaguement des étiquettes à chaque scène.

Les chercheurs, Patrik Vuilleumier et Timothée Proix, ont décidé d'utiliser divers modèles d'apprentissage automatique pour extraire le plus de caractéristiques possibles de ces fragments filmiques. D'où le but de notre projet.

Initialement, le projet était principalement une recherche visant à trouver les meilleurs modèles d'apprentissage automatique pour l'analyse des vidéos (en commençant par les modèles de détection d'objets) et à appliquer ces modèles aux fragments de films prédéfinis. Après quelques semaines, l'objectif du projet a évolué pour devenir la création d'un outil permettant d'appliquer différents modèles de détection d'objets aux vidéos et de visualiser les résultats dans une interface qui synchronise les fragments filmiques avec les séries temporelles tracées pour chaque classe d'objet détectée.

1.2.2 Objectif principal

L'objectif principal du logiciel est de faciliter l'utilisation des modèles de détection d'objets et de tester leur efficacité dans le cadre de cette étude spécifique sur l'émotion. Ce logiciel résoudra les problèmes en automatisant l'analyse détaillée des fragments de films et en fournissant des résultats analytiques facilement comparables aux données de l'activité cérébrale.

1.2.3 Solutions existantes

Les solutions existantes pour l'analyse de vidéos et la détection d'objets incluent diverses bibliothèques et outils d'apprentissage automatique, tels que PyTorch, OpenCV, et des modèles de détection d'objets comme YOLO (You Only Look Once). Cependant, ces outils nécessitent souvent une expertise technique pour être mis en œuvre et ne sont pas spécifiquement conçus pour être intégrés dans une interface utilisateur conviviale permettant une analyse synchronisée avec des données d'activité cérébrale. La solution proposée par notre

logiciel se distingue par sa capacité à intégrer facilement des modèles de détection d'objets dans une interface utilisateur, simplifiant ainsi le processus d'analyse pour les chercheurs.

1.2.4 Licence

Le code sera livré sous la licence MIT.

1.3 Liste des besoins

Les besoins des clients au début concernaient principalement une approche analytique pour extraire le maximum de données possibles sur une liste prédéfinie de films et les représenter sous forme de séries temporelles.

1.3.1 Besoins explicités par le client

- Analyser les fragments de films pour extraire des données détaillées sur les caractéristiques présentes dans chaque scène.
- Représenter les données extraites sous forme de séries temporelles statiques pour une analyse comparative avec les données d'activité cérébrale.
- Utiliser des modèles de détection d'objets pour identifier et classifier les éléments présents dans les scènes des films.

1.3.2 Besoins proposés par notre équipe

- Développer une interface utilisateur permettant la synchronisation des fragments de films avec une analyse en temps réel des caractéristiques détectées.
- Intégrer la possibilité de tester différents modèles de détection d'objets pour évaluer leur efficacité et leurs limitations dans le cadre de l'étude.
- Automatiser le processus de traitement des vidéos et de visualisation des résultats, facilitant ainsi l'analyse et la comparaison des données.
- Ajouter des fonctionnalités de recherche et de filtrage pour permettre aux utilisateurs de trouver facilement les classes d'objets détectées et leur correspondance dans le dataset COCO.

2 Développement

2.1 Conception

2.1.1 Scénario d'utilisation

L'application développée offre deux modes principaux : le mode traitement et le mode visualisation.

Mode Traitement(Processing Mode)

1. L'utilisateur télécharge une vidéo à traiter via l'interface.
2. L'utilisateur sélectionne le modèle de détection d'objets à utiliser parmi une liste de modèles pré-entraînés YOLO.
3. L'application traite la vidéo, en détectant les objets dans chaque cadre de la vidéo.
4. Deux fichiers sont générés : une vidéo annotée et un fichier JSON contenant les données de détection.
5. Une barre de progression indique l'avancement du traitement.

Mode Visualisation (Visualisation Mode)

1. L'utilisateur télécharge une vidéo annotée et un fichier JSON générés par le mode traitement.
2. L'utilisateur peut visualiser la vidéo annotée et les graphes synchronisés représentant les niveaux de confiance des détections au fil du temps.
3. L'utilisateur peut sélectionner différentes classes d'objets pour mettre à jour les graphes en temps réel.
4. En appuyant sur le bouton "Play", la vidéo et les graphes sont initiés simultanément.
5. L'utilisateur peut exporter les graphes pour une analyse plus approfondie.

2.1.2 Contraintes

- Le programme doit être capable de traiter différents formats de vidéo.
- Il doit être compatible avec plusieurs modèles de détection d'objets.
- L'interface doit être intuitive et facile à utiliser.
- Les résultats doivent être affichés en temps réel avec une synchronisation précise avec les vidéos.
- Les graphes doivent être capables d'être exportés et analysés séparément.

2.1.3 Méthodes de conception

Pour concevoir l'application, plusieurs méthodes ont été utilisées, notamment la création de diagrammes de classe et de séquence pour visualiser la structure et le flux du programme.

Diagramme de Classe Ce diagramme de classe montre la structure du programme, incluant la classe principale et tout les methodes implémentée.

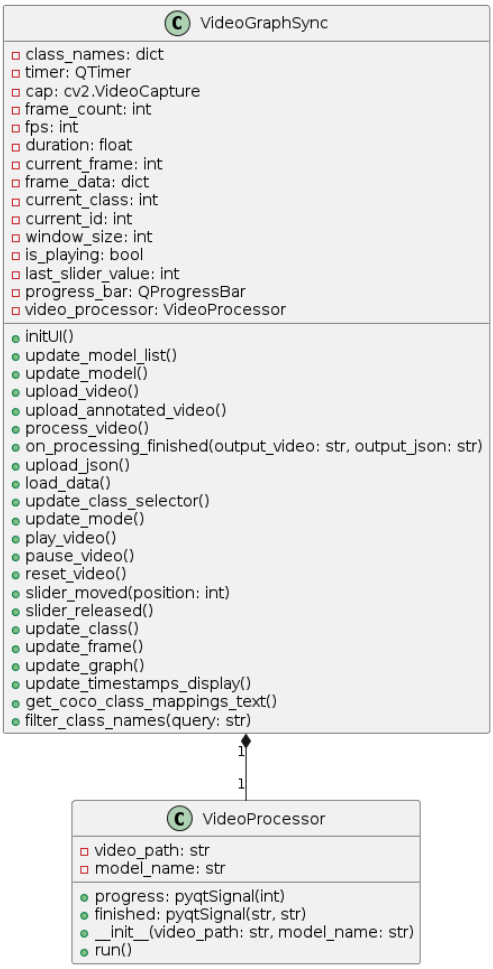

Figure 1: Diagramme de Classe

Diagramme de Cas d'Utilisation Ce diagramme illustre les principales interactions entre l'utilisateur et le système, mettant en évidence les différents cas d'utilisation.

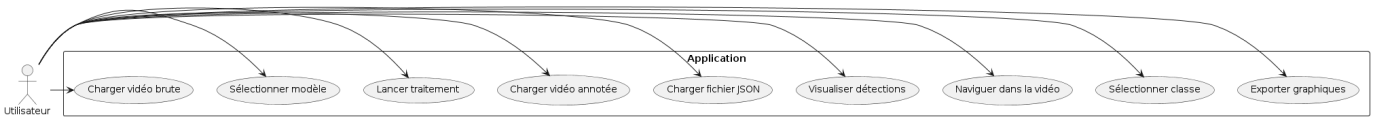

Figure 2: Diagramme de Cas d'Utilisation

Diagrammes de Séquence Ces diagrammes de séquence détaillent les processus de traitement vidéo et de visualisation, montrant les étapes clés et les interactions entre les composants.

Diagramme de Séquence pour le Traitement Vidéo Ce diagramme de séquence détaille le processus de traitement vidéo, depuis le téléchargement de la vidéo par l'utilisateur jusqu'à la génération des

fichiers de sortie.

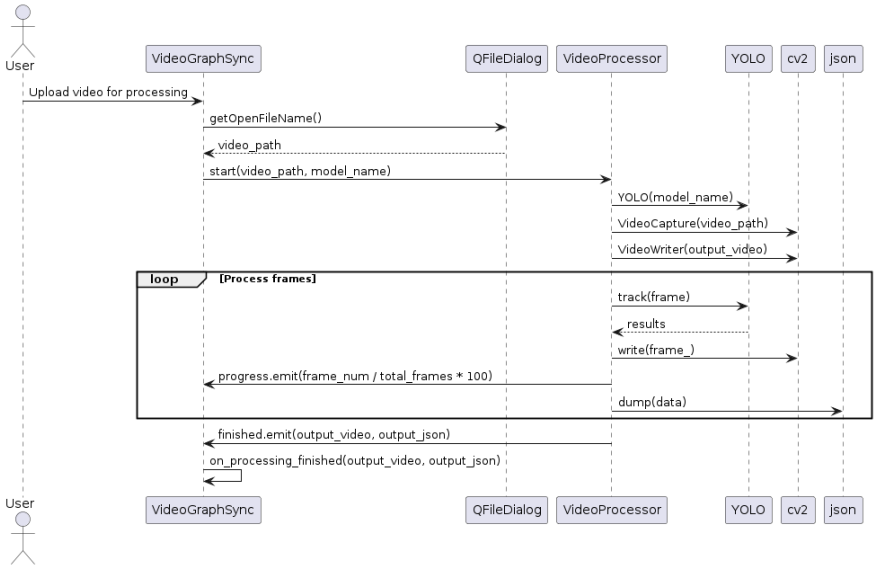

Figure 3: Diagramme de Séquence pour le Traitement Vidéo

Diagramme de Séquence pour la Visualisation Ce diagramme de séquence décrit le flux de travail pour la visualisation, y compris le téléchargement de la vidéo annotée et du fichier JSON, et la mise à jour des graphes en temps réel.

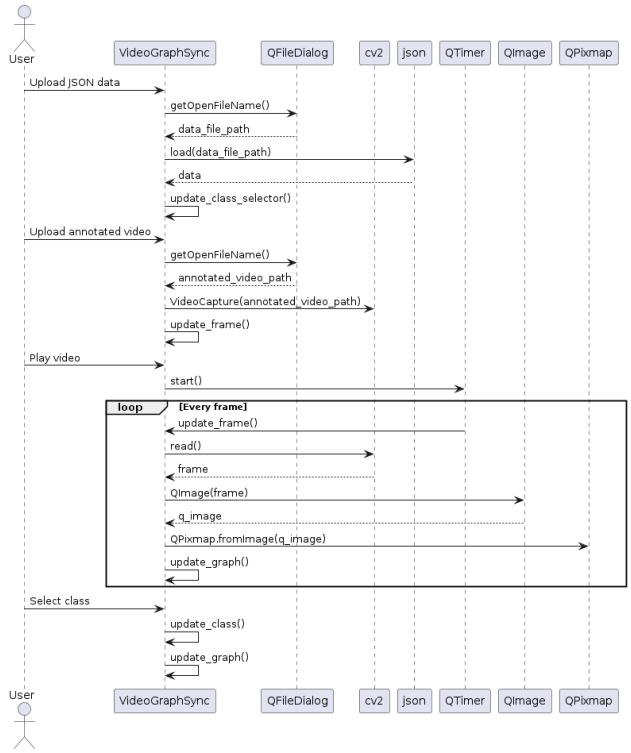

Figure 4: Diagramme de Séquence pour la Visualisation

2.2 Interface

L'interface utilisateur a été conçue pour être intuitive, permettant aux utilisateurs de télécharger des vidéos, de sélectionner des modèles de détection, et de visualiser les résultats de manière synchronisée.

Cas d'utilisation de l'interface

Téléchargement et traitement d'une vidéo

- L'utilisateur sélectionne une vidéo à traiter.
- L'interface affiche le chemin de la vidéo téléchargée et permet de choisir un modèle de détection d'objets.
- Pendant le traitement, une barre de progression indique l'avancement du processus.

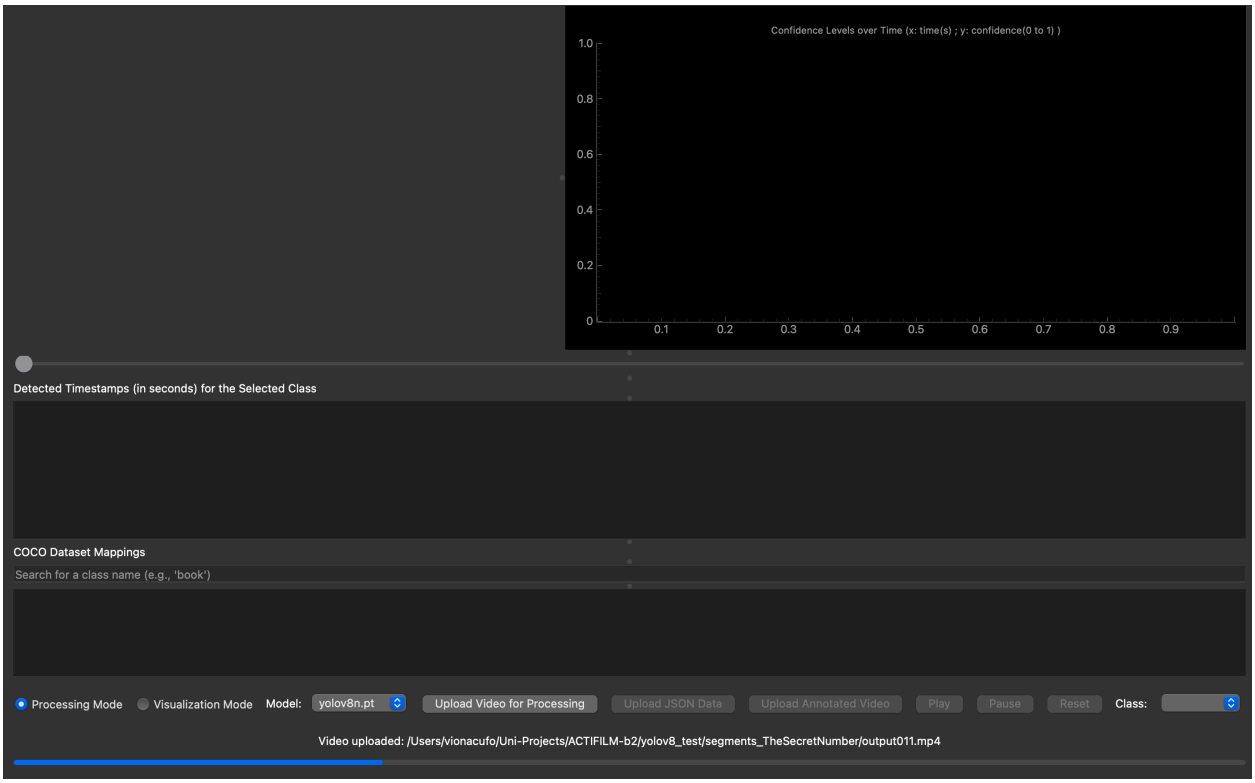

Figure 5: Téléchargement et traitement d'une vidéo

Téléchargement et visualisation de la vidéo annotée et du fichier JSON

- L'utilisateur sélectionne une vidéo annotée et un fichier JSON pour les visualiser.
- L'interface affiche les chemins des fichiers téléchargés.
- L'utilisateur peut lire la vidéo annotée et voir les graphes se mettre à jour en temps réel.
- Les graphes montrent les niveaux de confiance des détections en fonction du temps.

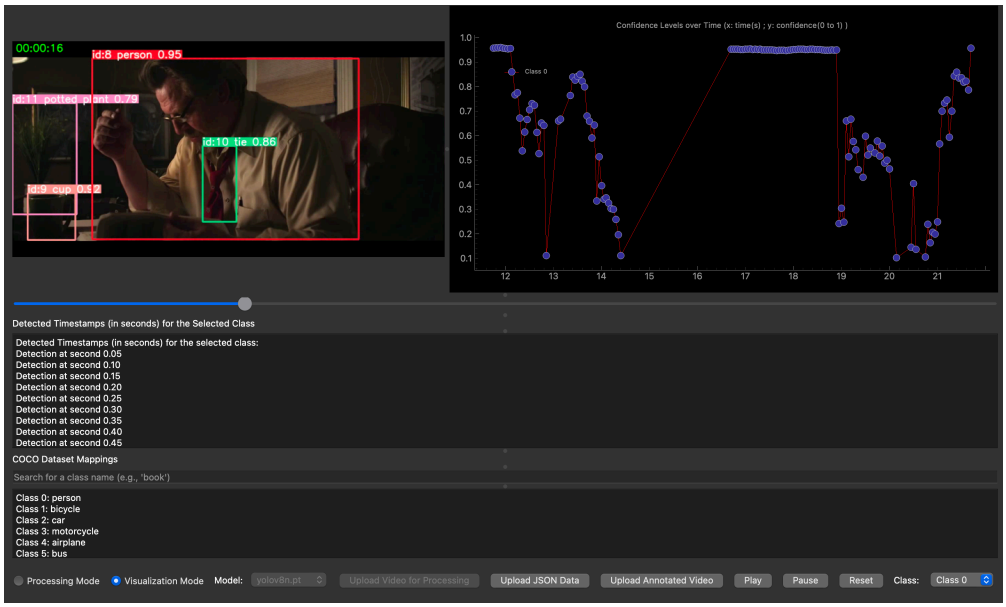

Figure 6: Téléchargement et visualisation de la vidéo annotée et du fichier JSON

Sélection et recherche de classes d'objets

- L'utilisateur peut sélectionner différentes classes d'objets pour mettre à jour les graphes.
- Les graphes et les horodatages affichent les détections pour la classe sélectionnée.
- L'utilisateur peut rechercher des classes d'objets dans le dataset COCO pour voir leur correspondance avec les numéros de classe.

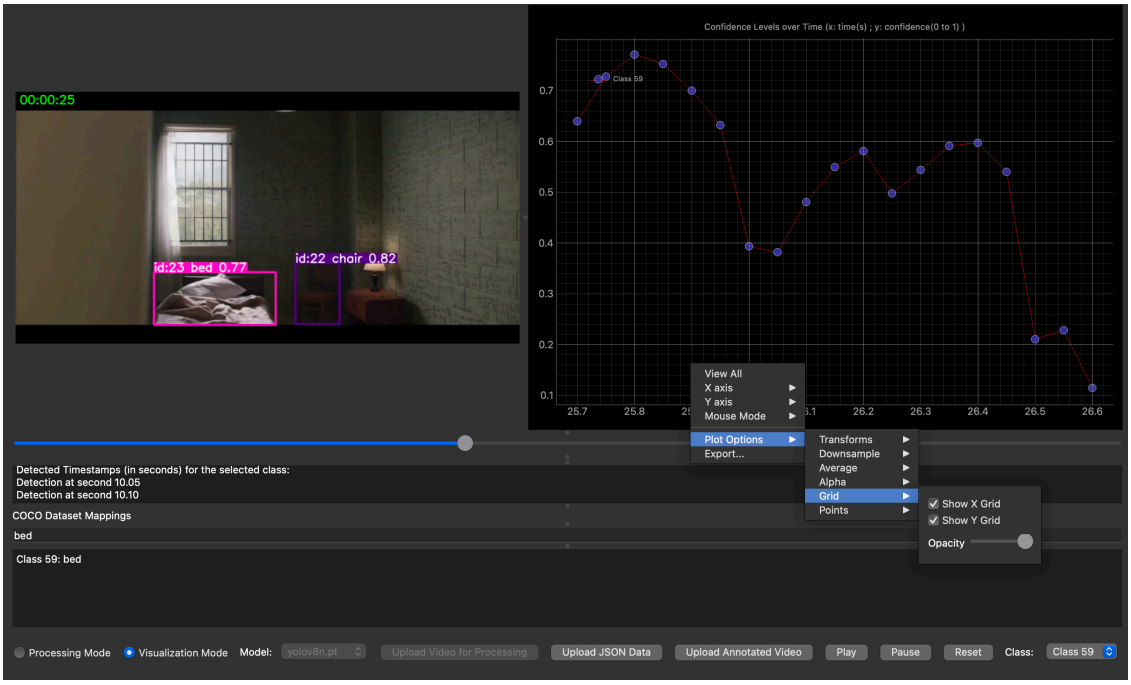

Figure 7: Sélection et recherche de classes d'objets

Voici les principaux éléments de l'interface :

- Boutons de Téléchargement: Pour télécharger des vidéos et des fichiers JSON.
- Sélecteur de Modèle: Pour choisir parmi une liste de modèles de détection d'objets.
- Barre de Progression: Pour indiquer l'avancement du traitement vidéo.
- Zone d'Affichage Vidéo: Pour visualiser les vidéos.
- Graphes Synchronisés: Pour afficher les niveaux de confiance des détections en fonction du temps.
- Boutons Play, Pause, Reset: Pour gérer l'avancée de la vidéo et du graphe au cours du temps
- Sélecteur de Classe: Pour choisir la classe d'objet à visualiser dans les graphes.
- Barre de Recherche: Pour rechercher des classes d'objets dans le dataset COCO.

2.3 Implémentation

2.3.1 Choix lors de l'implémentation

Lors de l'implémentation de notre projet, plusieurs choix ont dû être réalisés :

- **Modèles de détection d'objets**: Nous avons choisi d'utiliser les modèles YOLO (You Only Look Once) de la bibliothèque Ultralytics. Cela est dû à leur performance reconnue dans le domaine de la détection d'objets en temps réel et leur capacité à traiter des vidéos rapidement.
- **Interface utilisateur**: Pour l'interface utilisateur, nous avons opté pour PyQt5 combiné avec PyQtGraph. PyQt5 est une bibliothèque robuste et bien documentée pour la création d'interfaces graphiques, tandis que PyQtGraph est efficace pour le tracé de graphiques en temps réel.
- **Traitement vidéo**: OpenCV a été sélectionné pour le traitement vidéo en raison de sa flexibilité et de son large liste de fonctionnalités pour la manipulation des vidéos.
- **Format des fichiers de sortie**: Nous avons décidé de générer des fichiers JSON pour stocker les résultats des détections d'objets et des vidéos annotées. Le format JSON est facilement lisible et manipulable, ce qui facilite l'analyse des données.

2.3.2 Outils informatiques à disposition

Pour réaliser ce projet, nous avions plusieurs outils informatiques à notre disposition :

- **Environnements de développement intégrés (IDE)**: PyCharm, Visual Studio Code, etc.
- **Systèmes de contrôle de version**: Git, GitHub, GitLab.
- **Bibliothèques de traitement vidéo**: OpenCV, FFmpeg.
- **Frameworks de développement d'interface utilisateur**: Tkinter, Streamlit, PyQt, Kivy.
- **Outils de visualisation de données**: Matplotlib, Seaborn, PyQtGraph.
- **Services de cloud computing**: Google Colab, AWS, Azure.

2.3.3 Outils informatiques utilisés

Pour notre projet, nous avons utilisé les outils suivants :

- **Langage de programmation**: Python
- **Environnement de développement**: Visual Studio Code
- **Système de contrôle de version**: Git.
- **Bibliothèques et frameworks**:
  - **OpenCV**: Pour le traitement des vidéos.
  - **PyQt5**: Pour le développement de l'interface utilisateur.
  - **PyQtGraph**: Pour la visualisation des graphes en temps réel.
  - **Ultralytics**: Pour les modèles de détection d'objets YOLO.
  - **NumPy**: Pour les opérations numériques et le traitement des données.
  - **JSON**: Pour le stockage et la manipulation des résultats des détections d'objets.
- **Services de cloud computing**: Google Colab. Utilisé pour exécuter des tests pour des traitements nécessitant des ressources GPU, ce qui a permis de réduire le temps de traitement des vidéos.

2.3.4 Justification des choix d'outils

Les choix des outils ont été guidés par les critères suivants :

- **Performance et efficacité**: Les modèles YOLO et OpenCV sont reconnus pour leur performance et leur rapidité, ce qui est crucial pour le traitement de vidéos en temps réel.
- **Simplicité et facilité d'utilisation**: PyQt5 et PyQtGraph permettent de créer des interfaces graphiques intuitives et interactives.
- **Flexibilité et extensibilité**: Python et ses bibliothèques offrent une grande flexibilité pour le développement et l'extension du projet.
- **Accès aux ressources GPU**: Google Colab permet d'accéder facilement à des ressources GPU pour accélérer le traitement des vidéos.

2.4 Tests et évaluation

2.4.1 Méthodes de test

Pour garantir le bon fonctionnement du programme, nous avons utilisé plusieurs méthodes de test :

Tests d'intégration L'intégration de différents modules a été testée pour s'assurer qu'ils fonctionnent correctement ensemble. Cela inclut l'interaction entre l'interface utilisateur (PyQt5), le traitement vidéo (OpenCV) et les modèles de détection d'objets (Ultralytics YOLO).

Les tests d'intégration ont été réalisés en exécutant des scénarios d'utilisation complets, où une vidéo est chargée, traitée et visualisée dans l'interface.

Tests fonctionnels Les fonctionnalités de l'interface utilisateur ont été testées pour vérifier qu'elles répondent aux besoins des utilisateurs finaux. Cela comprend le chargement de vidéos, la sélection de modèles, l'affichage des résultats en temps réel et la génération de graphes.

Des tests manuels ont été effectués pour s'assurer que l'interface est intuitive et que toutes les interactions se déroulent sans erreurs.

Tests de performance Les tests de performance ont été réalisés pour évaluer le temps de traitement des vidéos en fonction de leur longueur et de leur résolution. Les résultats sont les suivants :

- **Vidéo de 1 minute en 1080p** :
  - **Sur l'interface (CPU Intel Core i9)** :
    - Temps de traitement pour exécuter le modèle YOLO choisi: 5 jusqu'a 15 minutes
  - **Script additionnel sur Google Colab (T4 GPU)** :
    - Temps de traitement pour exécuter le modèle YOLO choisi : 2 minutes
    - Temps total pour le script (installation des bibliothèques, chargement de la vidéo, téléchargement du fichier JSON et de la vidéo annotée) : 4 minutes

Le script additionnel est disponible pour les utilisateurs n'ayant pas accès à un GPU. Ce script permet d'utiliser les GPU gratuits de Google Colab pour le traitement des vidéos plus longues. Une fois les fichiers (vidéo annotée et JSON) générés, ils peuvent être utilisés directement dans le mode de visualisation de l'interface.

2.4.2 Critères d'évaluation et de validation

Les critères suivants ont été utilisés pour évaluer et valider le logiciel :

Exactitude

- Les données de détection fournies par les modèles d'apprentissage automatique doivent être correctement synchronisées avec les vidéos.
- Les données JSON générées doivent contenir des informations précises sur les objets détectés, y compris leurs coordonnées et leur confiance, et doivent être correctement représentées sur les graphes.
- Les graphes doivent être mis à jour avec précision lorsqu'un utilisateur sélectionne une nouvelle classe.

Évaluation : Succès. Les résultats de détection sont bien synchronisés avec les vidéos et les graphes, et les classes peuvent être changées dynamiquement, affichant les données correctes.

Performance

- Le programme devait traiter les vidéos en temps réel ou avec un minimum de latence.
- L'interface utilisateur devait rester réactive même lors du traitement de grandes quantités de données.

Évaluation : Partiellement réussi. Le traitement des vidéos prend du temps en raison de l'utilisation de modèles de détection d'objets basés sur l'apprentissage automatique, qui sont optimisés pour fonctionner sur des GPU. Lorsqu'ils sont exécutés sur des CPU, ces modèles peuvent nécessiter des temps de traitement plus longs. Pour résoudre ce problème, nous proposons un script supplémentaire pour exécuter le traitement vidéo sur Google Colab (dans le cas où l'utilisateur n'a pas accès à un GPU), permettant d'utiliser des GPU gratuits pour accélérer le traitement.

Facilité d'utilisation

- L'interface devait être intuitive et facile à naviguer pour les utilisateurs finaux.
- Les utilisateurs devaient pouvoir facilement charger des vidéos, sélectionner des modèles et visualiser les résultats.

Évaluation : Succès partiel. L'interface est généralement facile à utiliser, mais des améliorations peuvent être apportées, comme l'affichage direct des noms des classes dans le menu déroulant.

Robustesse

- Le programme devait être stable et capable de gérer les erreurs de manière correcte, sans erreurs.
- Les tests ont inclus des cas de bord, comme le chargement de vidéos de différents formats et résolutions, pour vérifier la robustesse du programme.

Évaluation : Succès partiel. Le programme gère bien de nombreuses erreurs, mais des tests supplémentaires sont nécessaires pour évaluer la robustesse face à d'autres types d'erreurs momentanément inconnues.

2.4.3 Pistes d'amélioration

Bien que le logiciel soit fonctionnel et ait satisfait les critères d'évaluation, plusieurs améliorations peuvent encore être apportées :

- **Extension des modèles de détection** :
  - Ajouter la possibilité d'utiliser d'autres modèles de détection d'objets et de classification, ainsi que des mises à jour régulières des modèles existants.
- **Amélioration de l'interface utilisateur** :
  - Ajouter plus de fonctionnalités pour personnaliser l'affichage des graphes et des vidéos, comme le zoom et la sélection de plages de temps spécifiques.
  - Afficher directement les noms des classes dans le menu déroulant pour faciliter la sélection.
- **Détection des IDs pour chaque classe:**
  - Une amélioration potentielle serait la détection des IDs uniques pour chaque classe d'objet afin de distinguer différents objets de la même classe. Bien que cette fonctionnalité ait été tentée, elle n'a pas été pleinement implémentée en raison des inconsistances dans le fichier JSON produit par le modèle YOLO, où les IDs étaient parfois manquants pour certaines classes.

3 Organisation

3.1 Division du travail

3.1.1 Client et équipe de développement

Nous avons principalement travaillé ensemble en effectuant des tests sur l'ordinateur d'une personne et en recherchant les outils et méthodes à utiliser sur un autre. Le client a fourni des retours réguliers et des orientations.

3.1.2 Réunions

Nous avons eu des réunions hebdomadaires avec le client pour discuter de l'avancement et définir les prochaines étapes. Nous avons également eu trois réunions avec notre superviseur : une en début de semestre en personne, une avec les deux clients et une dernière pour présenter notre travail.

3.1.3 Suivi du travail

Pour chaque rendez-vous hebdomadaire, nous avons pris des notes des sujets discutés, des pistes à explorer, des possibilités de développement de notre programme ainsi que des liens et articles à lire.

3.2 Travail en groupe

3.2.1 Partage des tâches

En binôme, nous avons alloué 5-6 heures par semaine pour travailler ensemble, principalement pour définir les étapes pratiques, les logiciels à utiliser et anticiper les problèmes potentiels afin de répondre aux demandes du client. Pendant ces sessions, nous avons souvent travaillé sur les mêmes tâches pour mieux comprendre et trouver les meilleures façons de les accomplir. Cependant, une fois les méthodes et outils déterminés, nous avons essayé de nous occuper de tâches différentes pour maximiser notre efficacité. Malgré cela, il y avait parfois des redondances dans notre travail en raison de l'incertitude initiale sur la meilleure manière de réaliser certaines tâches. Nous avons ensuite partagé nos progrès et ajusté nos stratégies lors de nos réunions conjointes et des rendez-vous avec le client.

3.2.2 Volume horaire

Nous avons consacré environ 100 heures de travail à ce projet depuis le début, sans compter les rendez-vous.

4 Conclusion

La documentation complète et le code source du projet se trouvent dans le dépôt GitLab du projet. Vous y trouverez des instructions détaillées pour l'installation, l'utilisation et les contributions futures au projet.