Université Hassan 1er

Faculté des Sciences et Techniques

-Settat-

RAPPORT DE PROJET

MASTER SCIENCES ET TECHNIQUES

Spécialité : Réseau et système informatiques

Smart City Vision

CLOUD COMPUTING SERVICES

Réalisé par :
Sous la direction de :

IBNOUDOUKKALI ANAS — M. EZZATI ABDELAZIZ

LAARIFI MOHAME

Année Universitaire : 2025-2026

Sommaire

Introduction Générale

Le besoin croissant de solutions pour les villes intelligentes a conduit au développement de systèmes intelligents visant à optimiser la gestion urbaine et améliorer la qualité de vie des citoyens. L'un de ces systèmes est le Système d'Éclairage Intelligent (SEI), conçu pour améliorer l'efficacité énergétique et la sécurité publique en contrôlant l'éclairage public en fonction de la présence de personnes ou de véhicules. Ce projet explore l'intégration de l'intelligence artificielle (IA) et de l'informatique en nuage pour améliorer les fonctionnalités du SEI.

La première partie de cette pratique consiste à capturer des flux vidéo en temps réel à partir de caméras publiques pour surveiller des zones de la ville. Ces flux sont traités localement pour identifier les objets, spécifiquement les personnes et les véhicules, afin de contrôler le système d'éclairage intelligent. Grâce à l'utilisation du modèle d'IA Ultralytics YOLOv8, l'objectif est de compter et de suivre ces objets de manière efficace.

Dans la deuxième phase, le traitement est externalisé vers le cloud pour explorer les avantages de la scalabilité et de l'optimisation des performances. La charge de traitement est transférée vers une machine virtuelle (VM) dans le cloud, permettant une meilleure gestion des ressources et la capacité de gérer des volumes de données plus importants.

De plus, le projet explore la gestion de plusieurs caméras pour couvrir une zone plus vaste, et examine l'utilisation des services d'IA dans le cloud, tels que Azure Machine Learning, pour améliorer les capacités du système. Enfin, des options de calcul sans serveur seront explorées pour rationaliser davantage le processus et réduire les coûts opérationnels.

Ce rapport décrit les étapes suivies tout au long du projet, les défis rencontrés, les décisions prises pour les résoudre et les métriques de performance collectées lors de l'implémentation de chaque tâche. L'objectif est non seulement de créer un prototype fonctionnel, mais aussi d'explorer le potentiel des services d'IA basés sur le cloud dans les applications des villes intelligentes.

CHAPITRE I ARCHITECTURE DU SYSTEME

Introduction

Le projet est basé sur une architecture **client-serveur** permettant de traiter des images provenant de flux vidéo en temps réel. Cette approche divise le traitement entre un client léger, qui capture et envoie les images, et un serveur, qui effectue l'analyse intensive à l'aide du modèle YOLOv8. Voici une explication détaillée de l'architecture et des technologies utilisées.

1. Composants de l'architecture

L'architecture est composée de deux principaux composants :

- ***Client :***
  - Le client est une application Python exécutée localement sur une machine qui capture les images depuis une caméra publique accessible via une URL. Il utilise OpenCV pour gérer le flux vidéo en continu.
  - Les images sont capturées à intervalles réguliers (toutes les 5 secondes), sérialisées avec Pickle, puis envoyées au serveur via une connexion socket TCP.
  - Le client peut également mettre en pause la capture ou ajuster le zoom selon les besoins.
- ***Serveur :***
  - Le serveur est une application Python hébergée sur une machine capable de supporter le traitement d'images en temps réel. Il utilise le modèle YOLOv8, fourni par la bibliothèque Ultralytics YOLO, pour détecter les personnes dans les images reçues.
  - Le serveur désérialise les données envoyées par le client, applique l'algorithme de détection, puis renvoie le nombre de personnes détectées au client via la connexion socket.
  - Le serveur gère également plusieurs connexions clients et peut traiter simultanément des images provenant de plusieurs flux vidéo.

2. Technologies Utilisées

P le **ython** est un Langage de programmation principal pour le client et serveur, choisi pour sa simplicité et ses nombreuses bibliothèques de traitement d'images.

Figure 1 : logo de python

OpenCV est un Bibliothèque utilisée pour capturer et manipuler des images depuis le flux vidéo. Elle permet d'extraire des images en temps réel depuis une URL de caméra publique.

Figure 2 : logo de OpenCV

Streamlit est une bibliothèque Python permettant de créer rapidement des applications web interactives pour l'analyse et la visualisation de données, avec une syntaxe simple et intuitive.

Figure 3 : logo de Streamlit

YOLOv8 (Ultralytics YOLO) : Modèle d'intelligence artificielle utilisé pour la détection d'objets. YOLOv8 est performant pour la détection en temps réel et a été choisi pour sa précision et sa rapidité.

Pickle : Utilisé pour sérialiser et désérialiser les données (images), facilitant leur envoi via la connexion socket.

Sockets TCP : Protocoles de communication permettant l'échange de données entre le client et le serveur. Le socket TCP assure une transmission fiable des images et des résultats.

3. Avantages de l'Architecture

Traitement Déporté : L'architecture client-serveur permet de décharger le client du traitement intensif des images, en l'envoyant au serveur. Cela permet au client d'être léger et de fonctionner sur des appareils avec des ressources limitées.

Scalabilité : Le serveur peut être hébergé sur une machine puissante ou sur le cloud, ce qui permet de traiter simultanément plusieurs flux vidéo provenant de différents clients.

Modularité : Le système est conçu de manière modulaire, facilitant l'ajout de nouvelles fonctionnalités, telles que la détection d'autres objets ou l'analyse avancée des données.

4. Conclusion

En conclusion, cette architecture client-serveur permet un traitement efficace et distribué des images en temps réel, avec un serveur dédié à l'analyse des données à l'aide de YOLOv8. Le système est modulaire et scalable, offrant une gestion optimale des flux vidéo. Dans la prochaine partie, nous aborderons le **traitement local** des images pour améliorer la détection des objets et visualiser les résultats via une interface interactive.

CHAPITRE II TRAITEMENT LOCAL

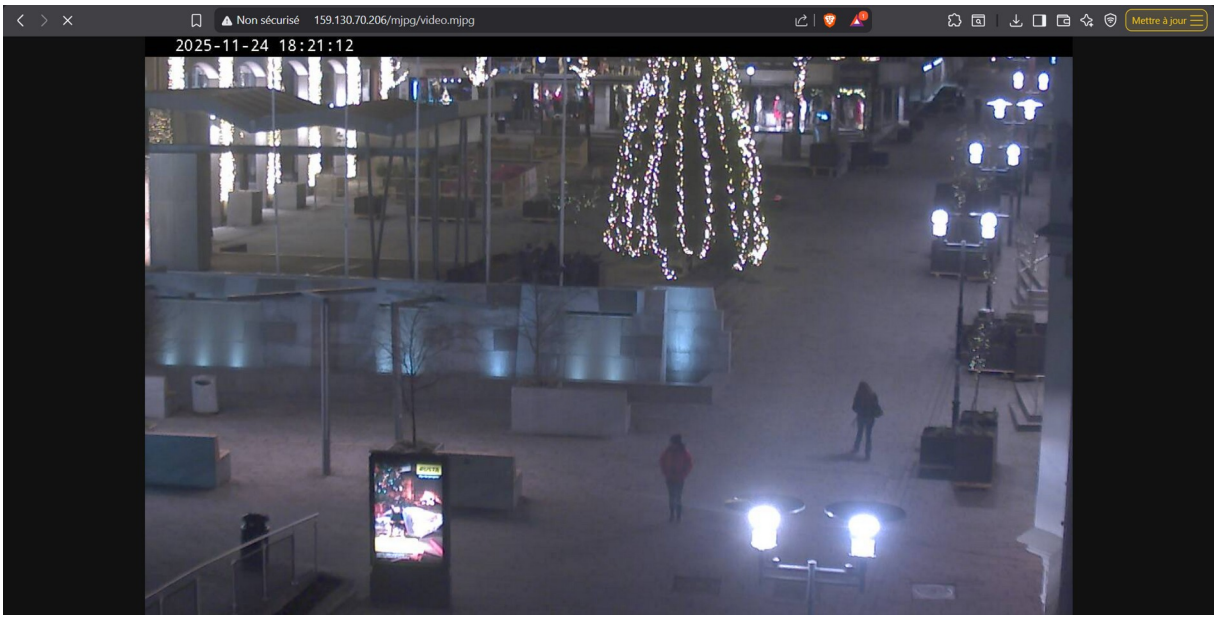
Introduction

Dans cette première partie du projet, l’objectif principal est de capturer des images à partir d'une webcam publique et de les traiter localement pour détecter des objets, comme des personnes et des véhicules. Ce processus est crucial pour alimenter le système d'éclairage intelligent en fonction de la présence des éléments dans la zone surveillée. Nous avons utilisé une webcam publique en streaming et un modèle d'intelligence artificielle (IA) basé sur **YOLOv8** pour la détection des objets.

1. Identification d'une webcam publique

Pour réaliser cette détection, nous avons utilisé des webcams publiques diffusant un flux vidéo en temps réel de différentes zones urbaines. Ces caméras sont accessibles via Internet et permettent d'observer la circulation des personnes et des véhicules. Nous avons choisi des caméras fixes, offrant une vue stable et continue de la rue.

Voici une capture d'écran d'une webcam choisie dans ce projet :



2. Capturer une image de la webcam

Après avoir sélectionné la webcam publique, nous avons mis en place un système pour capturer une image toutes les 5 secondes à partir du flux vidéo en temps réel. Nous avons utilisé un script Python pour automatiser cette tâche. La résolution de l'image capturée doit être suffisamment élevée pour permettre la détection des objets (personnes ou véhicules).

3. Détection des personnes dans les images

Pour détecter les personnes présentes dans les images capturées, nous avons utilisé le modèle **YOLOv8** (You Only Look Once), qui est un modèle de détection d'objets en temps réel. YOLOv8 permet de détecter de manière efficace et précise les objets dans des images, en l’occurrence, les personnes et les véhicules.

Le code suivant applique le modèle YOLOv8 à l’image capturée pour détecter les personnes et dessiner des boîtes autour d'elles :

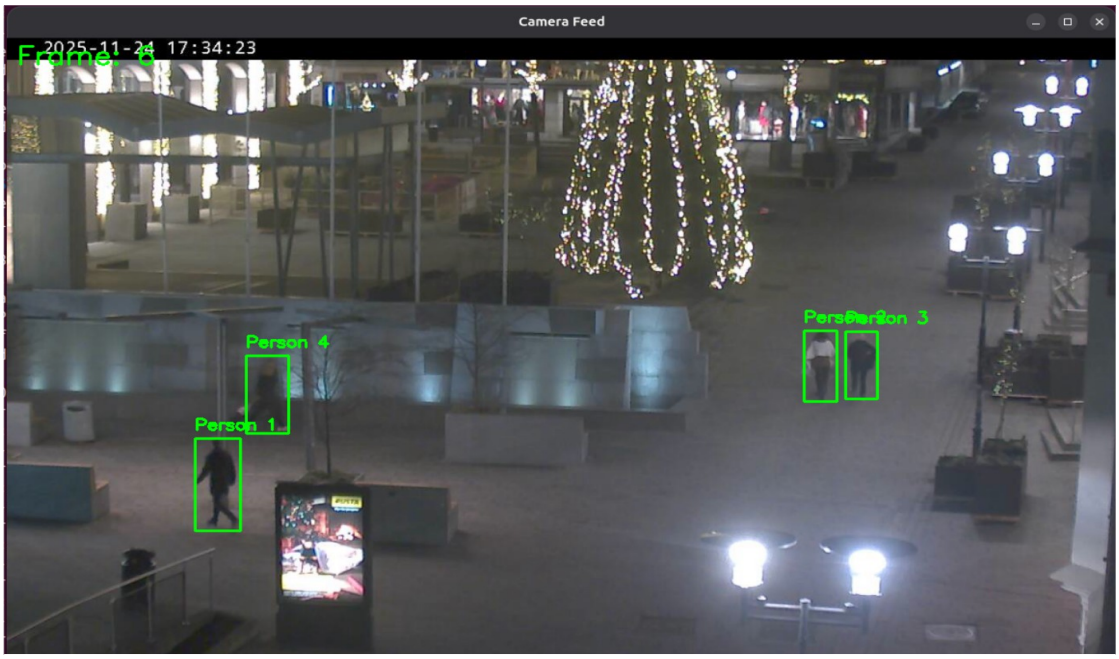
```python
# Run detection with error handling
try:
    detections = yolo_net.predict(source=result, verbose=False)

    # Check if detections exist and have boxes
    if detections and len(detections) > 0 and detections[0].boxes is not None:
        names_map = getattr(detections[0], "names", None) or getattr(yolo_net, "names", {})
        for box in detections[0].boxes:
            try:
                class_id = int(box.cls.item())
                confidence = float(box.conf.item())
                class_name = names_map.get(class_id, f"class_{class_id}") if isinstance(names_map, dict) else str(class_id)

                if confidence > 0.3:
                    detected_objects[class_name] += 1

                # Check if the detected object is a person (class_id 0)
                if class_id == 0 and confidence > 0.5:
                    person_count = person_count + 1
            except Exception as e:
                print(f"[WARNING] Error processing detection box: {e}")
                continue
except Exception as e:
    print(f"[ERROR] Detection failed: {e}")
    import traceback
    traceback.print_exc()
    # Continue with person_count = 0
```

Voici un exemple de detection :



3.1 Résultats de la détection

Le nombre de personnes détectées dans chaque image est enregistré dans un fichier CSV, avec la date et l'heure de la capture. Voici un exemple du fichier CSV généré :

3.2

people_count_camera1.csv

```
date,people_count
2025-11-20 00:00:00,2
2025-11-20 00:00:05,0
2025-11-20 00:00:10,0
2025-11-20 00:00:15,0
2025-11-20 00:00:20,2
2025-11-20 00:00:25,0
2025-11-20 00:00:30,0
2025-11-20 00:00:35,0
2025-11-20 00:00:40,0
2025-11-20 00:00:45,0
2025-11-20 00:00:50,0
2025-11-20 00:00:55,0
2025-11-20 00:01:00,0
2025-11-20 00:01:05,0
2025-11-20 00:01:10,0
2025-11-20 00:01:15,0
2025-11-20 00:01:20,0
2025-11-20 00:01:25,2
2025-11-20 00:01:30,1
2025-11-20 00:01:35,1
2025-11-20 00:01:40,2
2025-11-20 00:01:45,0
2025-11-20 00:01:50,0
2025-11-20 00:01:55,0
```

Graphique des résultats

Les résultats obtenus à partir du comptage des personnes détectées ont été enregistrés dans un fichier **CSV**, qui contient les informations suivantes : la **date** de la capture et le **nombre de personnes détectées**. Ces données ont été utilisées pour générer un graphique illustrant l'évolution du nombre de personnes détectées au fil du temps.

3.2.1 Visualisation des données via Streamlit

Pour rendre les résultats plus accessibles et interactifs, nous avons utilisé un fichier Python appelé **people_count_dashboard.py**, qui utilise la bibliothèque **Streamlit**. Ce fichier permet de transformer le fichier **people_count.csv** en une interface web dynamique et interactive. Grâce à cette interface, l'utilisateur peut visualiser le graphique des résultats tout en ayant la possibilité d'interagir avec les données.

Le fichier **people_count_dashboard.py** charge les données à partir du fichier CSV et génère un graphique montrant l'évolution du nombre de personnes détectées au fil du temps. L'interface permet également d'appliquer des filtres de dates à l'aide d'un curseur interactif, afin de sélectionner une période spécifique et observer les variations du nombre de personnes détectées dans cet intervalle.

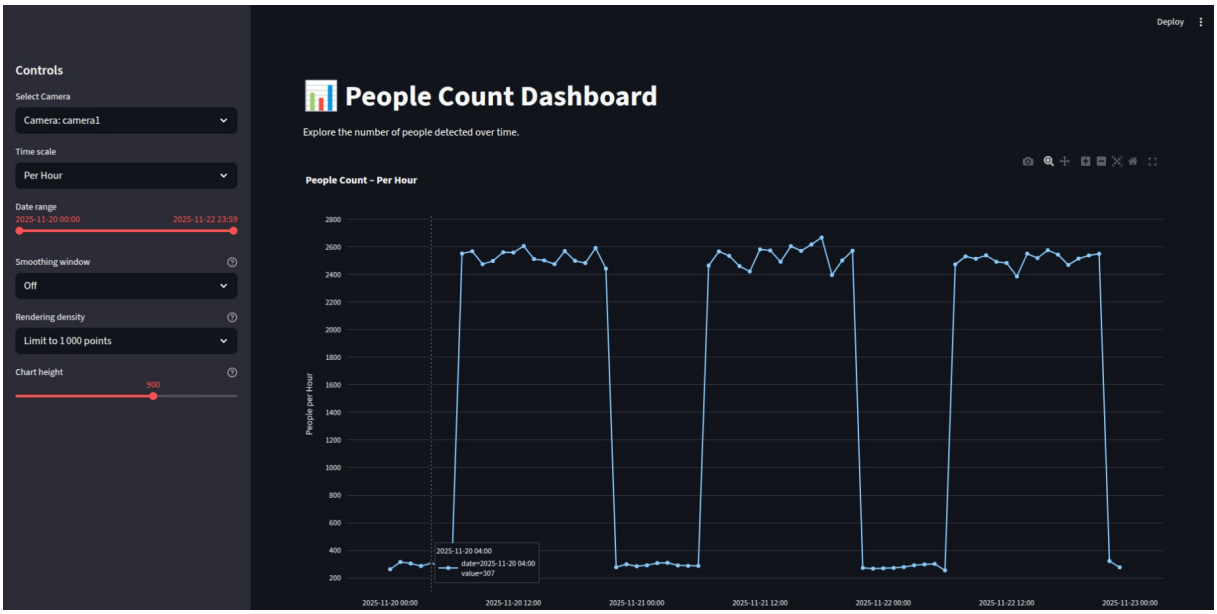
Voici un extrait du code utilisé dans **people_count_dashboard.py** pour générer un graphique interactif avec **Streamlit** :

```
fig = px.line(
    resampled,
    x="date",
    y="value",
    markers=True,
    title=f"People Count – {scale_label}",
)
# Determine tick format based on time scale
if scale_label == "Per 5 Seconds":
    # Show date, time, and seconds for 5-second intervals
    tickformat = "%Y-%m-%d %H:%M:%S"
elif scale_label in ["Per Minute", "Per Hour"]:
    # Show date and time for minutes and hours
    tickformat = "%Y-%m-%d %H:%M"
else:
    # Show only date for days
    tickformat = "%Y-%m-%d"

fig.update_layout(
    xaxis_title="Time",
    yaxis_title=y_label,
    hovermode="x unified",
    xaxis=dict(
        rangeslider=dict(visible=True),
        tickformat=tickformat,
        type="date",
    ),
    yaxis=dict(
        tickmode="auto",
        nticks=20,
    ),
    height=chart_height,
)

st.plotly_chart(
    fig,
    use_container_width=True,
    config={"displaylogo": False, "scrollZoom": True},
)
```

Voici un Exemple du graphique interactif :



Conclusion

La première partie du projet a permis de mettre en place un système local efficace pour capturer et traiter les images, ainsi que pour détecter les personnes et véhicules à l'aide de l'IA. Grâce à l'interface interactive créée avec **Streamlit**, nous avons pu visualiser et analyser les résultats de manière intuitive. Dans la **Partie 2**, nous explorerons l'externalisation du traitement vers le cloud afin d'améliorer les performances et la scalabilité du système.

CHAPITRE III TRAITEMENT DANS LE CLOUD

Introduction

Dans cette deuxième partie du projet, l'objectif est d'externaliser le traitement des données vers le cloud, afin de profiter des avantages de scalabilité, de performance et de gestion des ressources. Nous avons utilisé des **machines virtuelles (VM)** dans le cloud, en particulier sur la plateforme **Microsoft Azure**, pour héberger et traiter les données de comptage des personnes détectées par le système local. Ce passage au cloud permet de décharger le système local et d'optimiser le traitement des données en utilisant des ressources cloud puissantes.

1. Création de la machine virtuelle (VM) dans le cloud

Pour cette étape, nous avons créé une machine virtuelle sur **Microsoft Azure**, utilisant la configuration **Standard B1s** recommandée. Cette machine virtuelle est utilisée pour exécuter le traitement des images capturées par les webcams publiques. L'utilisation d'une machine virtuelle permet de bénéficier de la flexibilité du cloud, avec la possibilité d'allouer des ressources de manière dynamique en fonction des besoins du projet.

Une fois la machine virtuelle créée, nous avons installé les bibliothèques nécessaires à l'exécution du traitement des images et de la détection des personnes.

2. Envoi des images vers la machine virtuelle

Une fois la machine virtuelle configurée, nous avons développé un script pour envoyer les images capturées localement vers la VM dans le cloud. Cela permet de traiter les données dans un environnement cloud, offrant plus de puissance de calcul et une gestion centralisée des ressources.

Voici un exemple de script Python qui envoie les images capturées depuis le système local vers la machine virtuelle dans le cloud :

```
# Send frame to server
try:
    result = pickle.dumps(frame)
    print(f"[INFO] Sending frame {frame_count} from {self.camera_name}...")
    client_socket.sendall(result)
    client_socket.sendall(str.encode("foto"))

    # Wait for response
    sleep(0.5)  # Give server time to process
    data = client_socket.recv(1024)

    if data:
        people_counter = int(data.decode('utf8'))
        print(f"[INFO] Frame {frame_count} - People detected: {people_counter}")
        print("-" * 80)
```

3. Traitement des données sur la machine virtuelle

Une fois les images envoyées vers la machine virtuelle, le traitement des données de détection des personnes est effectué à l'aide du modèle **YOLOv8**, tout comme dans la **Partie 1** du projet. Le modèle d'IA YOLOv8 est utilisé pour détecter les personnes dans les images envoyées.

4. Conclusion

Dans cette partie, nous avons externalisé le traitement des données vers le cloud en utilisant des machines virtuelles sur **Microsoft Azure**, ce qui a permis d'améliorer les performances et de gérer efficacement les ressources. Cette étape a préparé le système à la gestion de plusieurs caméras, qui sera abordée dans la **Partie 3**.

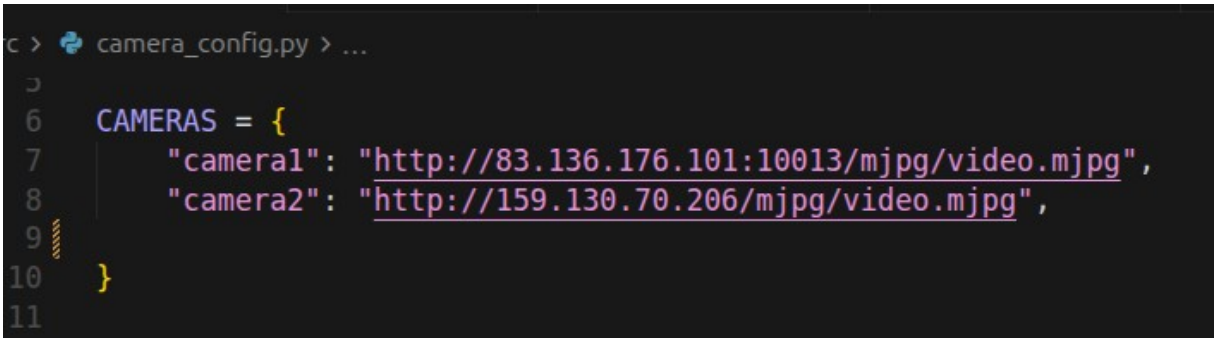
CHAPITRE IV GESTION MULTI- CAMERAS

Introduction

Le système multi-caméra permet de surveiller simultanément plusieurs emplacements à partir d'un seul serveur centralisé. Au lieu d'avoir un système séparé pour chaque caméra, toutes les caméras envoient leurs images vidéo vers un serveur unique qui effectue la détection et le comptage de personnes. Chaque caméra est identifiée par un nom unique (par exemple "camera1", "camera2", "entree_principale") et génère ses propres données de comptage dans un fichier CSV dédié. Cette approche offre plusieurs avantages : une gestion centralisée, une utilisation optimale des ressources computationnelles, et la possibilité d'ajouter facilement de nouvelles caméras sans modifier le code du serveur.

1. Configuration Simple

L'ajout d'une nouvelle caméra dans le système est extrêmement simple grâce à un fichier de configuration centralisé. Il suffit d'ajouter une ligne dans le fichier `camera_config.py` en associant un nom de caméra à son adresse (URL pour une caméra IP, ou numéro pour une webcam locale). Par exemple, pour ajouter une troisième caméra, on ajoute simplement `"camera3": "http://192.168.1.100/mjpg/video.mjpg"` dans la liste. Le système génère automatiquement le fichier de données correspondant (`people_count_camera3.csv`) dès que la caméra commence à envoyer des images. Cette simplicité permet de déployer rapidement un réseau de surveillance avec des dizaines de caméras sans complexité technique supplémentaire.



2. Isolation des Données

Chaque caméra génère son propre fichier CSV contenant ses données de comptage avec des horodatages précis. Par exemple, la caméra "camera1" crée le fichier `people_count_camera1.csv`, tandis que "camera2" crée `people_count_camera2.csv`. Cette séparation des données permet plusieurs choses importantes : l'analyse indépendante de chaque point de vue, la comparaison des patterns entre différentes caméras, et la maintenance facilitée (on peut archiver ou supprimer les données d'une caméra sans affecter les autres). De plus, en cas de problème avec une caméra, les données des autres caméras restent intactes et accessibles.

3. Visualisation dans Streamlit

Le dashboard Streamlit intègre automatiquement toutes les caméras configurées dans le système. Lorsqu'on ouvre le dashboard, un menu déroulant dans la barre latérale affiche la liste de toutes les caméras disponibles. L'utilisateur peut simplement sélectionner une caméra dans ce menu pour visualiser ses données de comptage sous forme de graphiques temporels. Si une nouvelle caméra est ajoutée dans le fichier de configuration, elle apparaît

automatiquement dans le menu sans nécessiter de modification du code du dashboard. Cette intégration transparente rend la visualisation des données multi-caméras aussi simple que la visualisation d'une seule caméra.

4. Avantages de la gestion multi-caméras

Réduction des coûts d'infrastructure : Le système peut couvrir plusieurs zones géographiques simultanément avec une seule plateforme de traitement.

Maintenance simplifiée : Toutes les caméras sont gérées centralement, ce qui permet un suivi facile de leur état.

Analyse comparative : Vous pouvez analyser différentes zones géographiques pour mieux comprendre les flux de personnes ou les zones les plus fréquentées.

Analyse granulaire et globale : Chaque caméra génère des données individuelles tout en permettant de les regrouper pour une vue d'ensemble plus large.

5. Conclusion

En conclusion, la gestion de plusieurs caméras dans le système permet de garantir une **surveillance continue** et **en temps réel** de plusieurs zones, ce qui augmente l'efficacité et la réactivité du système. Le traitement parallèle des flux vidéo par la **machine virtuelle** est un élément clé de cette approche, permettant d'analyser plusieurs caméras en même temps sans compromettre les performances.