UNIVERSITÉ HASSAN 1er

FACULTE DES SCIENCES ET TECHNIQUES

SETTAT

Rapport du projet : Détection Intelligete de Personnes en Temps Réel

Réalisé par :

Chouaib Yakine

Basma El kasimi

Encadre par :

Mr Zaati

1. Introduction

Avec la prolifération des technologies d'intelligence artificielle et de traitement d'images, la détection de personnes en temps réel est devenue un enjeu majeur pour des applications de surveillance. Ce projet vise à développer une solution automatisée de détection de personnes à l'aide de caméras publiques et des techniques de traitement d'image avec **OpenCV**. Les données capturées, telles que les positions et les horaires de détection, sont stockées dans un fichier CSV pour une analyse approfondie. Ces informations sont ensuite traitées et visualisées avec **Matplotlib**, permettant une compréhension claire des résultats.

L'intégration de cette solution avec **AWS** (Amazon Web Services) permet non seulement de bénéficier d'une infrastructure scalable mais aussi d'assurer une gestion optimale des données à grande échelle, tout en garantissant une disponibilité et une résilience maximales pour notre système de détection.

Objectifs du Projet :

Ce projet a pour but d'intégrer l'intelligence artificielle et le cloud computing pour la gestion d'un système d'éclairage intelligent. Les objectifs principaux sont :

→ **Externalisation du Traitement IA vers le Cloud**
Utiliser **AWS** pour déporter le traitement des vidéos des caméras publiques vers le cloud. Cela permet de tirer parti des ressources évolutives d'AWS, réduisant ainsi la charge locale des machines.

→ **Détection d'Objets en Temps Réel**
Développer un système qui analyse les flux vidéo en temps réel pour détecter la présence de personnes ou de véhicules, ajustant automatiquement l'éclairage en fonction de l'activité détectée.

→ **Évaluation des Performances**
Comparer les performances du système en local et sur le cloud, en termes de coût, de ressources consommées (CPU, mémoire, bande passante) et de temps de réponse, afin de déterminer les avantages de chaque solution.

→ **Intégration des Services IA d'AWS**
Utiliser des services comme Amazon Rekognition pour la reconnaissance et la classification

des objets dans les vidéos, offrant une solution d’analyse évolutive et performante grâce aux modèles pré-entrainés d'AWS.

Contexte et enjeux

Les caméras publiques sont utilisées pour la surveillance urbaine et la détection d'objets en temps réel. Le traitement des flux vidéo nécessite des ressources importantes, d'où l'utilisation d'AWS pour externaliser ce traitement. Amazon Rekognition analyse les vidéos pour détecter des personnes et des véhicules, optimisant ainsi la précision et la rapidité. Ce projet explore l'efficacité du cloud pour le traitement d'images en temps réel et compare les performances du traitement local et cloud.

2. Configuration de l'Infrastructure sur AWS

Dans cette partie, on va voir comment on a configuré notre instance sur AWS et installé un gestionnaire de volumes logiques (LVM) pour gérer efficacement le stockage sur le serveur.

→ **Création de l'Instance EC2 sur AWS**

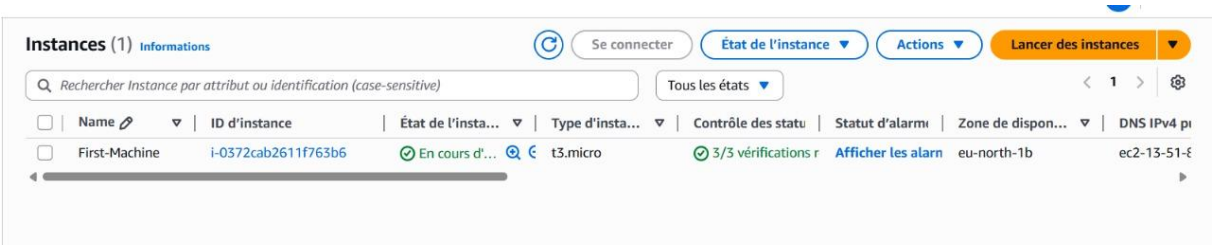
La première étape, bien sûr, est de créer une instance EC2 sur AWS. Voici comment on a procédé :

- **Connexion à la console AWS** :
  On commence par se connecter à la console AWS. Une fois connecté, on va dans "EC2" pour créer notre instance. L'interface est plutôt simple à suivre.
- **Lancer une nouvelle instance** :

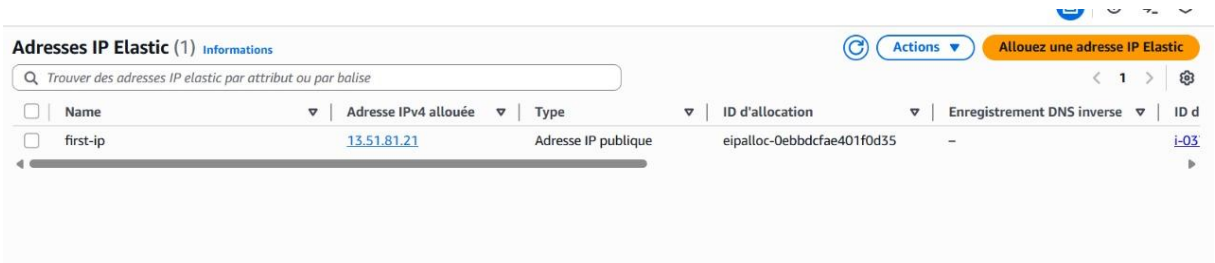
  On clique sur "Launch Instance".

  On choisit une image Ubuntu Server (par exemple, **Ubuntu 20.04 LTS**).

  Ensuite, on choisit une instance de type **t2.micro** (idéal pour commencer, surtout pour des projets légers).



- **Configurer le réseau et la sécurité** :
  On configure la connexion réseau, et surtout, on vérifie bien que le port **22** (SSH) est ouvert pour pouvoir se connecter à l'instance via SSH.
- **Obtenir l'IP publique** :
  Une fois l'instance lancée, on récupère l'IP publique. Cette adresse nous permet de nous connecter à l'instance en SSH.



→ **Connexion à l'Instance via SSH**

Une fois l'instance lancée, on utilise un terminal pour s'y connecter en SSH.

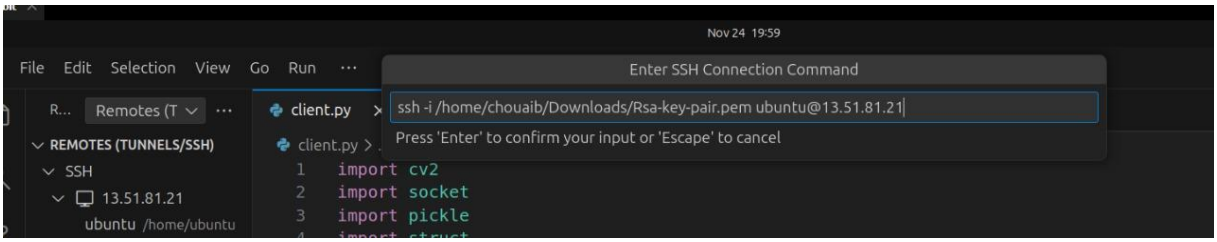
Cela permet d'accéder à l'instance pour pouvoir y installer et configurer les outils nécessaires.

3. Établissement de la Connexion Serveur-Client via AWS

Dans ce projet, nous avons mis en place un système de communication entre un serveur et un client en utilisant des sockets, afin de permettre une transmission fluide des données via un port réseau dédié. Voici les étapes principales de notre approche.

1. Connexion entre le client et le serveur

Une fois le serveur créé et configuré sur AWS EC2, la communication est établie à l'aide de sockets, qui assurent la transmission continue des flux vidéo. Le client se connecte au serveur via son adresse IP publique (grâce à une adresse IP Elastic dans AWS), ce qui permet au flux vidéo des caméras publiques d'atteindre le serveur pour le traitement.

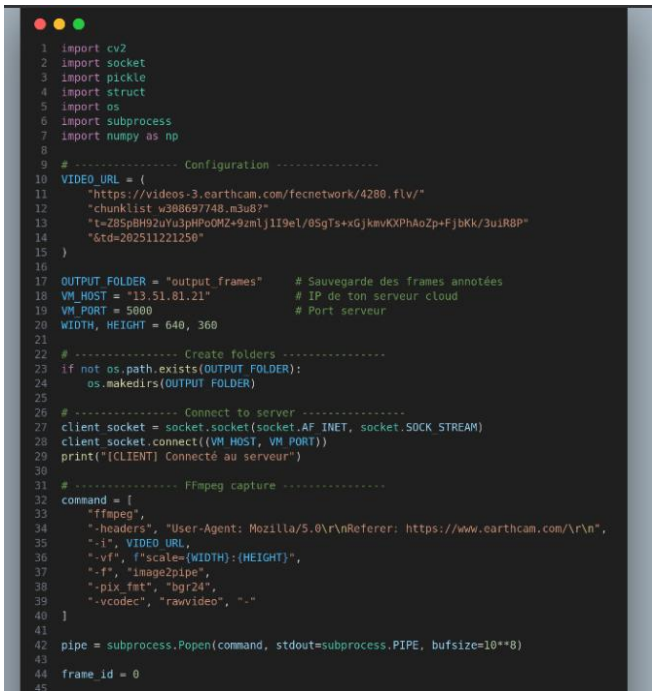


Voila la commande SSH du client pour se connecter au serveur AWS EC2 et envoyer les flux vidéo pour traitement en temps réel, avec une connexion sécurisée par clé privée.

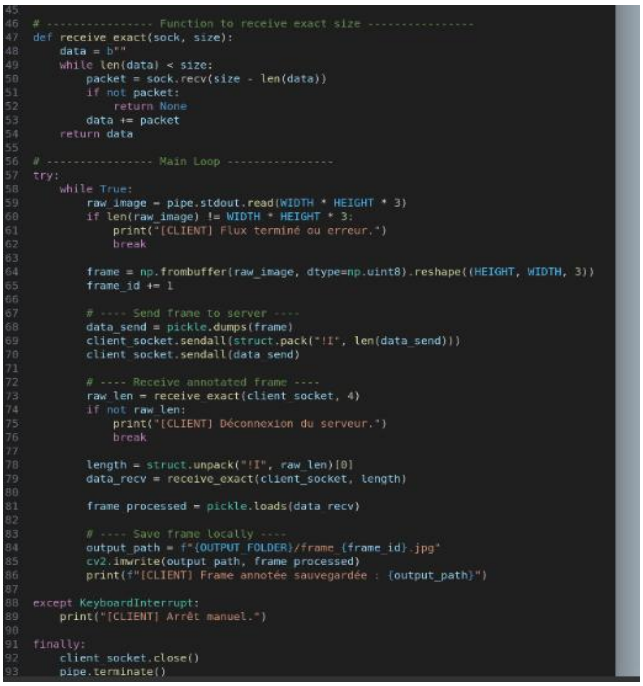
→ **Execution du code client :**

Du côté du client, nous avons exécuté un script Python nommé client.py. Ce script envoie des données à l'adresse IP publique du serveur (ou l'adresse locale en réseau privé), en utilisant le

même port que celui configuré sur le serveur. Une fois la connexion établie, les données envoyées sont reçues et traitées par le serveur grâce au script server.py.



```python
import cv2
import socket
import pickle
import struct
import os
import subprocess
import numpy as np

# ---------------- Configuration ----------------
VIDEO_URL = (
    "https://videos-3.earthcam.com/fecnetwork/4280.flv/"
    "chunklist_w308697748.m3u8?"
    "t=Z8SpBH92uYu3pHPoOMZ+9zmlj1I9el/0SgTs+xGjkmvKXPhAoZp+FjbKk/3uiR8P"
    "&td=202511221250"
)

OUTPUT_FOLDER = "output_frames"     # Sauvegarde des frames annotées
VM_HOST = "13.51.81.21"             # IP de ton serveur cloud
VM_PORT = 5000                      # Port serveur
WIDTH, HEIGHT = 640, 360

# ---------------- Create folders ----------------
if not os.path.exists(OUTPUT_FOLDER):
    os.makedirs(OUTPUT_FOLDER)

# ---------------- Connect to server ----------------
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((VM_HOST, VM_PORT))
print("[CLIENT] Connecté au serveur")

# ---------------- FFmpeg capture ----------------
command = [
    "ffmpeg",
    "-headers", "User-Agent: Mozilla/5.0\r\nReferer: https://www.earthcam.com/\r\n",
    "-i", VIDEO_URL,
    "-vf", f"scale={WIDTH}:{HEIGHT}",
    "-f", "image2pipe",
    "-pix_fmt", "bgr24",
    "-vcodec", "rawvideo", "-"
]

pipe = subprocess.Popen(command, stdout=subprocess.PIPE, bufsize=10**8)

frame_id = 0
```



```python

# ---------------- Function to receive exact size ----------------
def receive_exact(sock, size):
    data = b""
    while len(data) < size:
        packet = sock.recv(size - len(data))
        if not packet:
            return None
        data += packet
    return data

# ---------------- Main Loop ----------------
try:
    while True:
        raw_image = pipe.stdout.read(WIDTH * HEIGHT * 3)
        if len(raw_image) != WIDTH * HEIGHT * 3:
            print("[CLIENT] Flux terminé ou erreur.")
            break

        frame = np.frombuffer(raw_image, dtype=np.uint8).reshape((HEIGHT, WIDTH, 3))
        frame_id += 1

        # ---- Send frame to server ----
        data_send = pickle.dumps(frame)
        client_socket.sendall(struct.pack("!I", len(data_send)))
        client_socket.sendall(data_send)

        # ---- Receive annotated frame ----
        raw_len = receive_exact(client_socket, 4)
        if not raw_len:
            print("[CLIENT] Déconnexion du serveur.")
            break

        length = struct.unpack("!I", raw_len)[0]
        data_recv = receive_exact(client_socket, length)

        frame_processed = pickle.loads(data_recv)

        # ---- Save frame locally ----
        output_path = f"{OUTPUT_FOLDER}/frame_{frame_id}.jpg"
        cv2.imwrite(output_path, frame_processed)
        print(f"[CLIENT] Frame annotée sauvegardée : {output_path}")

except KeyboardInterrupt:
    print("[CLIENT] Arrêt manuel.")

finally:
    client_socket.close()
    pipe.terminate()
```

2. Configuration du serveur sur AWS EC2

Une fois la connexion établie, le serveur, qui est hébergé sur AWS EC2, reçoit les flux vidéo en continu. Le serveur doit être configuré pour écouter les connexions entrantes et traiter les données vidéo. Les groupes de sécurité dans AWS sont utilisés pour gérer l'accès au serveur en ouvrant les ports nécessaires (comme le port 5000 pour les connexions socket).

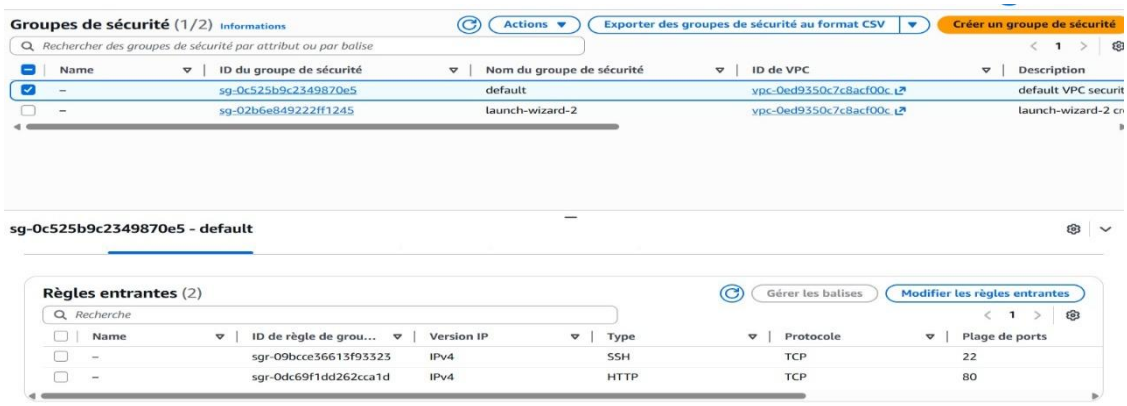
→ Execution du code serveur

Sur le serveur, nous avons lancé un script Python nommé server.py. Ce script est conçu pour écouter les connexions entrantes sur un port spécifique et traiter les données reçues. Il joue un rôle clé dans la gestion des requêtes envoyées par le client.

```python
import socket
import pickle
import struct
from ultralytics import YOLO
import cv2
import numpy as np
import csv
from datetime import datetime
import matplotlib.pyplot as plt
import pandas as pd

HOST = "0.0.0.0"
PORT = 5000

# Charger YOLO
yolo = YOLO("yolov8n.pt")

def receive_exact(sock, size):
    data = b""
    while len(data) < size:
        packet = sock.recv(size - len(data))
        if not packet:
            return None
        data += packet
    return data

def save_count_to_csv(count):
    with open("people_count.csv", "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([datetime.now().isoformat(), count])

def plot_people_count():
    try:
        df = pd.read_csv("people_count.csv", header=None, names=["timestamp", "count"])
        df["timestamp"] = pd.to_datetime(df["timestamp"])

        plt.figure(figsize=(10,5))
        plt.plot(df["timestamp"], df["count"], marker='o', linestyle='-')
        plt.xlabel("Temps")
        plt.ylabel("Nombre de personnes")
        plt.title("Nombre de personnes détectées au fil du temps")
        plt.grid(True)
        plt.tight_layout()

        # Sauvegarde en fichier PNG
        plt.savefig("people_count_graph.png")
        print("[GRAPH] Graphique sauvegardé : people_count_graph.png")

        # Optionnel : afficher si interface graphique disponible
        # plt.show()
```

3 . Sécurisation de la communication avec AWS

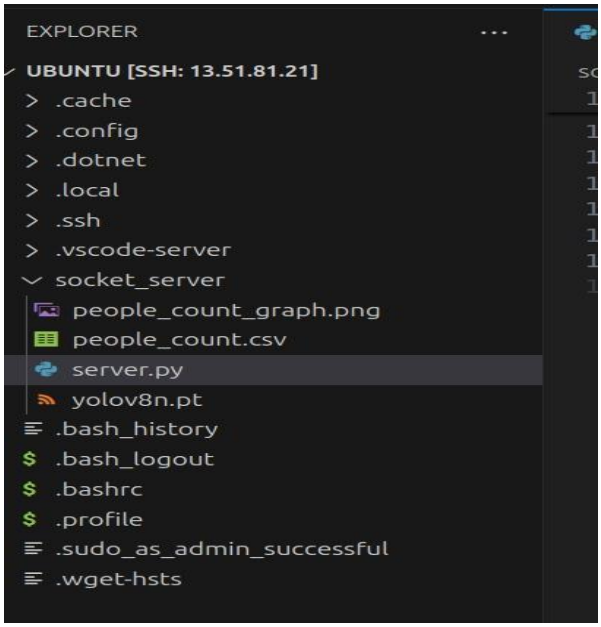
Pour sécuriser les données, on a configuré le serveur pour accepter seulement les connexions sur les ports et protocoles autorisés. Les groupes de sécurité AWS gèrent les accès et filtrent les connexions entrantes.



4. Déploiement de la solution sur AWS

Une fois le serveur et la connexion réseau configurés, le système peut être déployé sur AWS pour traiter les vidéos en temps réel. Le serveur EC2 est configuré pour démarrer le traitement dès qu'il reçoit les flux vidéo des caméras publiques.

→ **Structure du dossier du serveur**



Le dossier socket_server contient :

server.py - Script principal du serveur

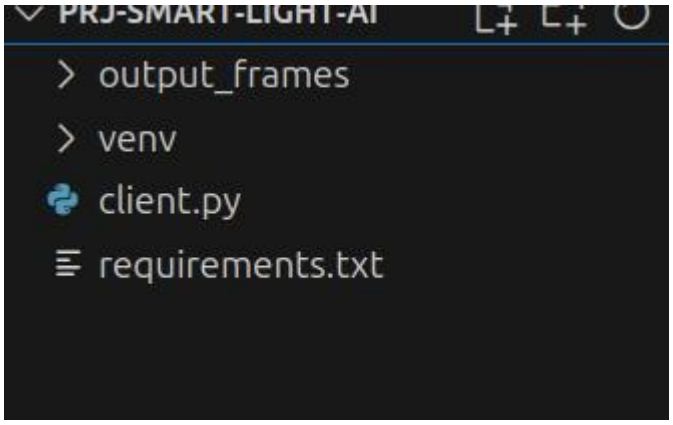
yolov8n.pt - Modèle YOLOv8 pour la détection d'objets

people_count.csv - Fichier des statistiques de détection

people_count_graph.png - Graphique visualisant les résultats

Configuration SSH réussie via VSCode sur l'IP 13.51.81.21, montrant une instance Ubuntu opérationnelle avec tous les fichiers essentiels déployés pour le système de détection en temps réel.

Structure du projet côté client :



client.py - Script principal pour la capture vidéo et l'envoi des données au serveur

requirements.txt - Fichier des dépendances Python nécessaires

output_frames/ - Dossier de stockage des images extraites du flux vidéo

venv/ - Environnement virtuel Python pour isoler les dépendances du projet

Cette organisation permet une exécution locale structurée du système de détection avant l'envoi des données vers le serveur AWS EC2.

4. Acquisition et Prétraitement des Images

Ce projet utilise le modèle **YOLOv8** pour détecter des personnes en temps réel à partir des flux vidéo capturés par des caméras publiques. Le système est conçu pour gérer les tâches suivantes :

- **Détection des personnes** : Le modèle YOLOv8 identifie et marque les personnes dans l'image.
- **Visualisation des détections** : Les images avec des rectangles autour des personnes détectées sont affichées à l'écran.
- **Enregistrement des données de détection** : Les informations concernant chaque détection (classe, confiance, coordonnées) sont organisées dans un DataFrame, facilitant l'analyse ou un traitement ultérieur.

Côté Client :

1. **Détection des Personnes :**
   Le client utilise le modèle **YOLOv8** pour détecter des personnes dans un flux vidéo en temps réel. Les cadres de délimitation sont dessinés autour des personnes détectées.
2. **Capture et Envoi des Visages :**
   Si une nouvelle personne est détectée ou si elle n'a pas été vue depuis 3 secondes, une image de son visage est enregistrée et envoyée au serveur via **socket**.

Côté Serveur :

Écoute des Connexions :

Le serveur écoute les connexions sur un port spécifié (par défaut 65432). Il reçoit d'abord la taille de l'image, puis l'image elle-même.

Validation et Sauvegarde :

Après validation de la taille, l'image est sauvegardée dans un répertoire **"**received_faces" avec un nom unique.

Enregistrement des Détections :

Un fichier **CSV** enregistre chaque détection avec un horodatage et les coordonnées de la détection. Le serveur garde aussi une trace des détections par minute.

Graphiques et Statistiques :

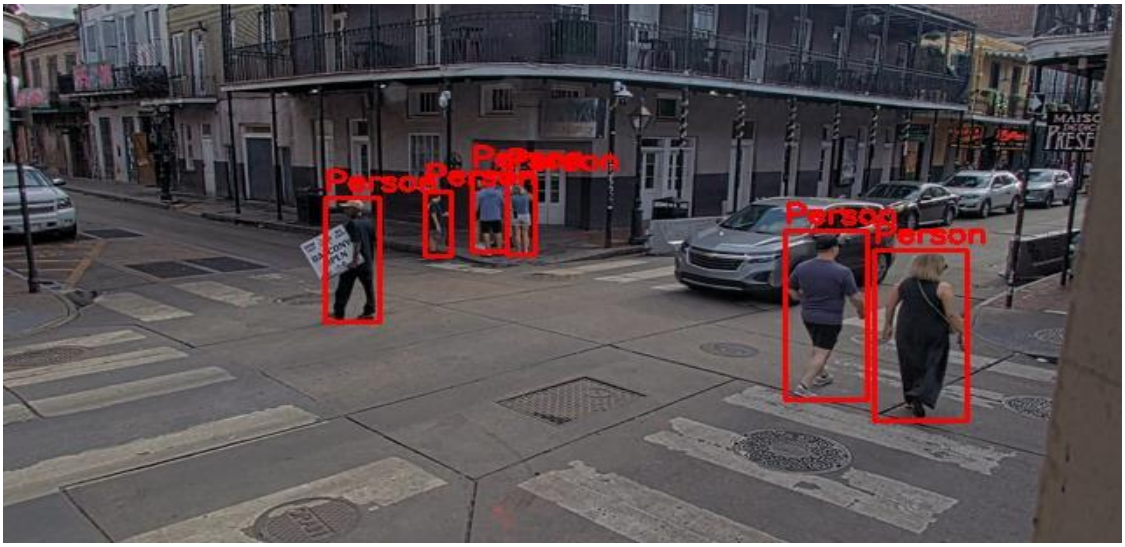
Un graphique des détections est généré et sauvegardé sous forme d'image pour un suivi visuel des statistiques.

5. Détection et Traitement des Données

Le traitement des données dans ce projet implique plusieurs étapes clés pour la détection des personnes à partir des flux vidéo. Le modèle YOLOv8 est utilisé pour effectuer la détection en temps réel. Voici les étapes principales :

1. Détection d'Objets dans les Flux Vidéo

On a utilisé le modèle **YOLOv8** pour détecter les personnes dans chaque image capturée à partir des flux vidéo. Une fois les personnes détectées, on a ajouté des rectangles autour d'elles pour les marquer sur l'image.



2. Filtrage et Identification des Nouvelles Détections

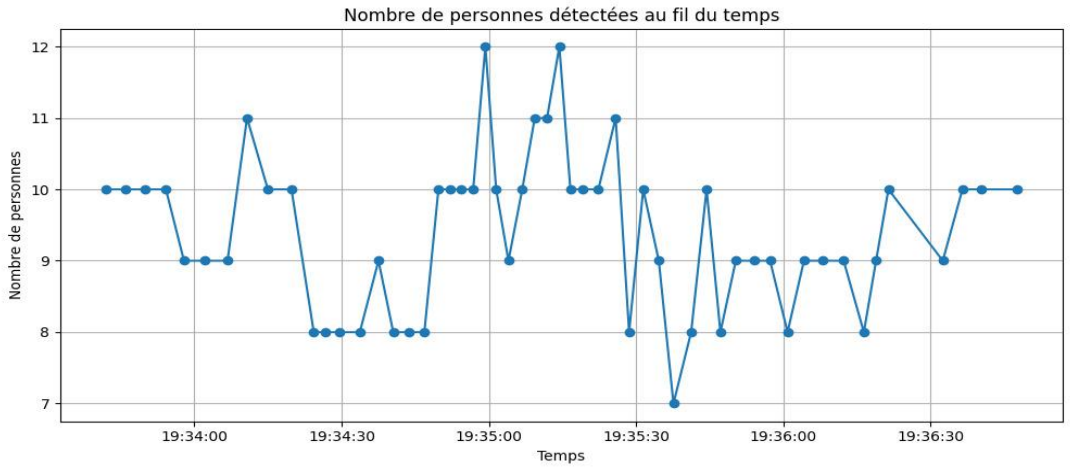
Les résultats sont filtrés pour ne conserver que les personnes détectées. Chaque nouvelle détection est comparée aux détections précédentes pour éviter les doublons. Si une personne n'a pas été détectée récemment (dans les 3 secondes), son visage est capturé et enregistré.

3. Sauvegarde et Transmission des Données

Les informations relatives à chaque détection (comme les coordonnées et l'heure de la détection) sont sauvegardées dans un fichier CSV pour une analyse ultérieure. Ces informations sont ensuite envoyées au serveur pour un stockage et un traitement supplémentaires.

4. Visualisation et Analyse

Les détections sont visualisées en temps réel à l'aide de Matplotlib, qui génère des graphiques sur le nombre de personnes détectées par minute. Cette étape permet d'avoir un suivi visuel des performances du système.



5.

6.

Résultats et Précision

Le modèle YOLOv8 a montré une précision élevée pour détecter les personnes en temps réel, et les résultats ont été enregistrés et traités de manière efficace grâce à la gestion de sockets pour l'échange de données entre le client et le serveur.

5. Technologies et Services AWS Utilisés

Python : On a utilisé Python comme langage principal pour développer les scripts de détection et de traitement des images. C’est un langage puissant et flexible, parfait pour ce type de projet.

OpenCV : a été utilisé pour capturer et traiter les images en temps réel. C’est un outil essentiel pour manipuler les flux vidéo et appliquer des techniques de traitement d'image.

YOLOv8 : Pour détecter les personnes dans les images, on a utilisé YOLOv8. Ce modèle de détection d'objets est super rapide et précis, ce qui permet de traiter les flux vidéo en temps réel sans latence.

Matplotlib : Une fois les détections effectuées, on a utilisé Matplotlib pour visualiser graphiquement les données de détection, comme le nombre de personnes détectées par minute. Cela permet de mieux comprendre les résultats du système.

Amazon EC2 : On a déployé notre serveur sur Amazon EC2. C'est une solution scalable qui nous permet de traiter les flux vidéo en temps réel en utilisant la puissance des serveurs dans le cloud.

Amazon S3 : Pour stocker les fichiers de données et les résultats de détection à grande échelle, on a utilisé Amazon S3. C'est une solution de stockage fiable et évolutive pour gérer de grandes quantités de données.

AWS Lambda : On a utilisé AWS Lambda pour exécuter certaines fonctions sans serveur. Cela permet de traiter les données détectées en temps réel sans avoir à gérer des serveurs, ce qui simplifie l'architecture du système.

AWS RDS : Enfin, AWS RDS a été utilisé pour gérer les bases de données des détections et stocker les résultats dans un environnement sécurisé. RDS nous aide à avoir une gestion fiable des données et une bonne sécurité.

2. Gestion avec Deux Caméras

Dans cette partie, on présente l'extension du système afin de gérer simultanément deux caméras, ce qui permet d'élargir la zone surveillée, d'améliorer la précision des détections et de rendre le système d'éclairage intelligent plus efficace.

.6. Architecture Multi-Caméras

Pour permettre l'utilisation de deux caméras en parallèle :

- Chaque caméra transmet son propre flux vidéo.
- Deux instances de YOLOv8 fonctionnent simultanément côté client (une par caméra).
- Chaque caméra est identifiée par un camera_id (par exemple : *cam1* et *cam2*).
- Les résultats des deux caméras sont traités séparément puis fusionnés par la suite.

Schéma logique simplifié :
Camera 1 → YOLOv8 → Traitement
Camera 2 → YOLOv8 → Traitement
→ Fusion des résultats → Stockage CSV + Visualisation

1. Communication Client–Serveur

Dans notre implémentation, nous avons mis en place une communication simple et efficace entre le client (qui gère les deux caméras) et le serveur AWS EC2.
Le système utilise un seul port et deux threads côté client, ce qui permet de transmettre simultanément les flux vidéo des deux caméras.

→ **Communication côté client**

Chaque caméra est traitée dans un thread indépendant :

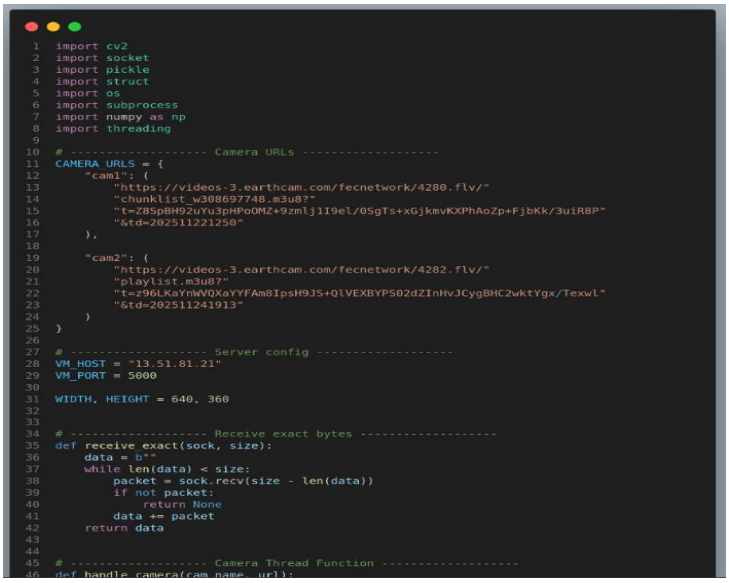
- Thread cam1 : capture et envoie les images de la première caméra
- Thread cam2 : capture et envoie les images de la deuxième caméra

Les deux threads exécutent la même fonction handle_camera(), mais chacun avec l'URL de sa caméra.

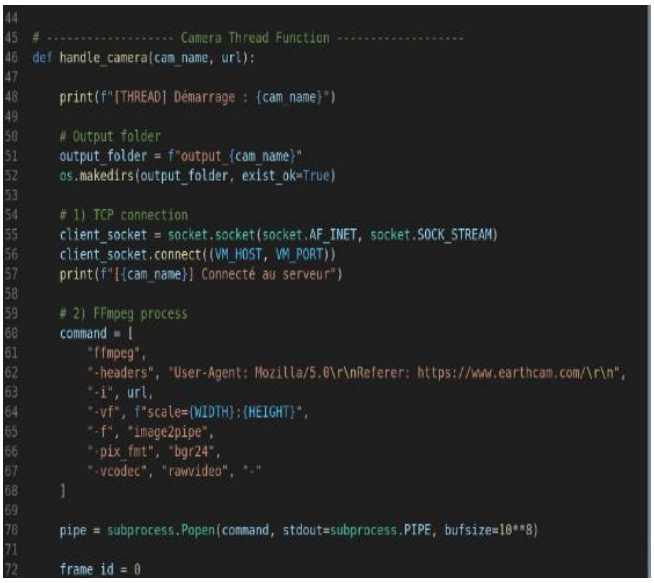
Pour chaque caméra :

1. FFmpeg récupère le flux vidéo public EarthCam.
2. FFmpeg convertit les images en format RAW (bgr24).

3. Les images sont envoyées au serveur via un socket TCP.
4. Nous transmettons aussi le nom de la caméra (ex : *cam1*, *cam2*) pour identifier la source.



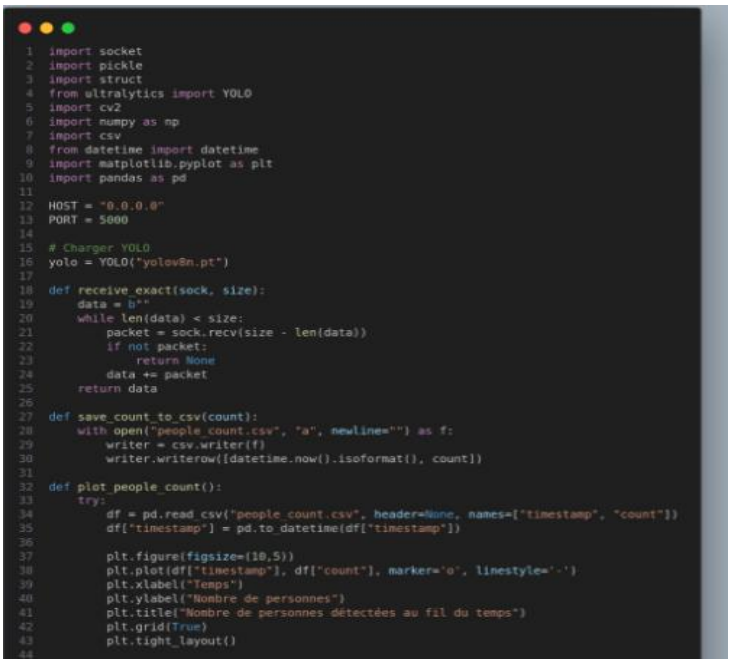
```python
import cv2
import socket
import pickle
import struct
import os
import subprocess
import numpy as np
import threading

# ------------------ Camera URLs ------------------
CAMERA_URLS = {
    "cam1": (
        "https://videos-3.earthcam.com/fecnetwork/4280.flv/"
        "chunklist_w308697748.m3u8?"
        "t=Z8SpBH92uYu3pHPoOMZ+9zmlj1I9el/0SgTs+xGjkmvKXPhAoZp+FjbKk/3uiR8P"
        "&td=202511221250"
    ),

    "cam2": (
        "https://videos-3.earthcam.com/fecnetwork/4282.flv/"
        "playlist.m3u8?"
        "t=z96LKaYnWVQXaYYFAm8IpsH9J5+QlVEXBYPS02dZInHvJCygBHC2wktYgx/Texwl"
        "&td=202511241913"
    )
}

# ------------------ Server config ------------------
VM_HOST = "13.51.81.21"
VM_PORT = 5000

WIDTH, HEIGHT = 640, 360


# ------------------ Receive exact bytes ------------------
def receive_exact(sock, size):
    data = b""
    while len(data) < size:
        packet = sock.recv(size - len(data))
        if not packet:
            return None
        data += packet
    return data


# ------------------ Camera Thread Function ------------------
def handle_camera(cam_name, url):
```



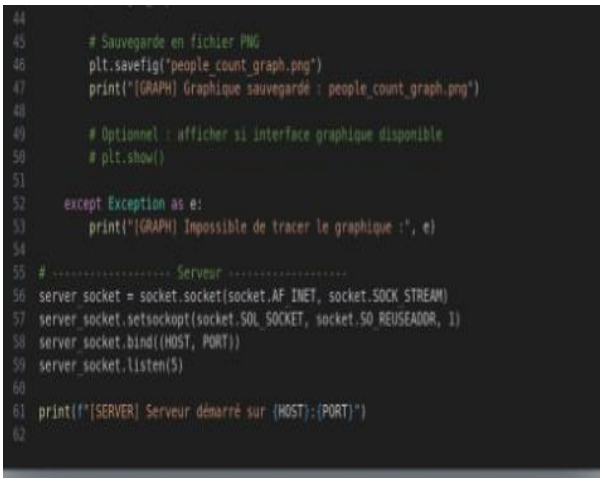
```python
# ------------------ Camera Thread Function ------------------
def handle_camera(cam_name, url):

    print(f"[THREAD] Démarrage : {cam_name}")

    # Output folder
    output_folder = f"output_{cam_name}"
    os.makedirs(output_folder, exist_ok=True)

    # 1) TCP connection
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((VM_HOST, VM_PORT))
    print(f"[{cam_name}] Connecté au serveur")

    # 2) FFmpeg process
    command = [
        "ffmpeg",
        "-headers", "User-Agent: Mozilla/5.0\r\nReferer: https://www.earthcam.com/\r\n",
        "-i", url,
        "-vf", f"scale={WIDTH}:{HEIGHT}",
        "-f", "image2pipe",
        "-pix_fmt", "bgr24",
        "-vcodec", "rawvideo", "-"
    ]

    pipe = subprocess.Popen(command, stdout=subprocess.PIPE, bufsize=10**8)

    frame_id = 0
```

Communication côté serveur AWS EC2

Extrait du code du serveur EC2



```python
import socket
import pickle
import struct
from ultralytics import YOLO
import cv2
import numpy as np
import csv
from datetime import datetime
import matplotlib.pyplot as plt
import pandas as pd

HOST = "0.0.0.0"
PORT = 5000

# Charger YOLO
yolo = YOLO("yolov8n.pt")

def receive_exact(sock, size):
    data = b""
    while len(data) < size:
        packet = sock.recv(size - len(data))
        if not packet:
            return None
        data += packet
    return data

def save_count_to_csv(count):
    with open("people_count.csv", "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([datetime.now().isoformat(), count])

def plot_people_count():
    try:
        df = pd.read_csv("people_count.csv", header=None, names=["timestamp", "count"])
        df["timestamp"] = pd.to_datetime(df["timestamp"])

        plt.figure(figsize=(10,5))
        plt.plot(df["timestamp"], df["count"], marker='o', linestyle='-')
        plt.xlabel("Temps")
        plt.ylabel("Nombre de personnes")
        plt.title("Nombre de personnes détectées au fil du temps")
        plt.grid(True)
        plt.tight_layout()
```



```python
        # Sauvegarde en fichier PNG
        plt.savefig("people_count_graph.png")
        print("[GRAPH] Graphique sauvegardé : people_count_graph.png")

        # Optionnel : afficher si interface graphique disponible
        # plt.show()

    except Exception as e:
        print("[GRAPH] Impossible de tracer le graphique :", e)

# ------------------ Serveur ------------------
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, PORT))
server_socket.listen(5)

print(f"[SERVER] Serveur démarré sur {HOST}:{PORT}")
```

Le serveur écoute sur un seul port (5000).
Il reçoit simultanément les données provenant des deux caméras grâce à deux connexions TCP distinctes, une par thread.

2. Synchronisation des Détections

Dans notre implémentation, les détections provenant des deux caméras sont centralisées et enregistrées dans un seul fichier CSV côté serveur.

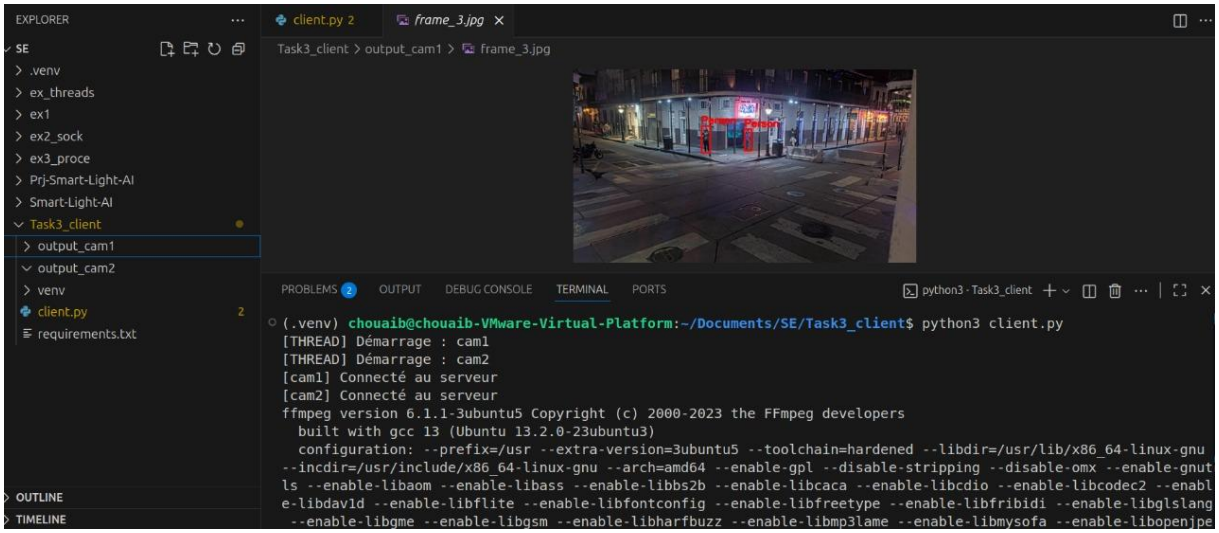
Chaque fois qu'une caméra envoie une image, le serveur ajoute une nouvelle ligne contenant toutes les informations nécessaires pour analyser l'activité en temps réel.

Chaque entrée du CSV inclut :

- l'heure exacte de la réception de l'image,
- les coordonnées de la détection (bounding box) obtenues par le client avant l'envoi,
- le niveau de confiance retourné par le modèle YOLOv8,
- la classe détectée (dans notre cas, principalement "person"),
- l'identifiant de la caméra (cam1 ou cam2), ce qui nous permet de savoir d'où vient chaque détection.

Cette fusion des données dans un seul fichier facilite l'analyse globale, la comparaison entre caméras, ainsi que la génération de graphiques et statistiques.

3. Visualisation des Résultats



Voici le résultat qui nous est sorti lors de l'exécution du système multi-caméras.

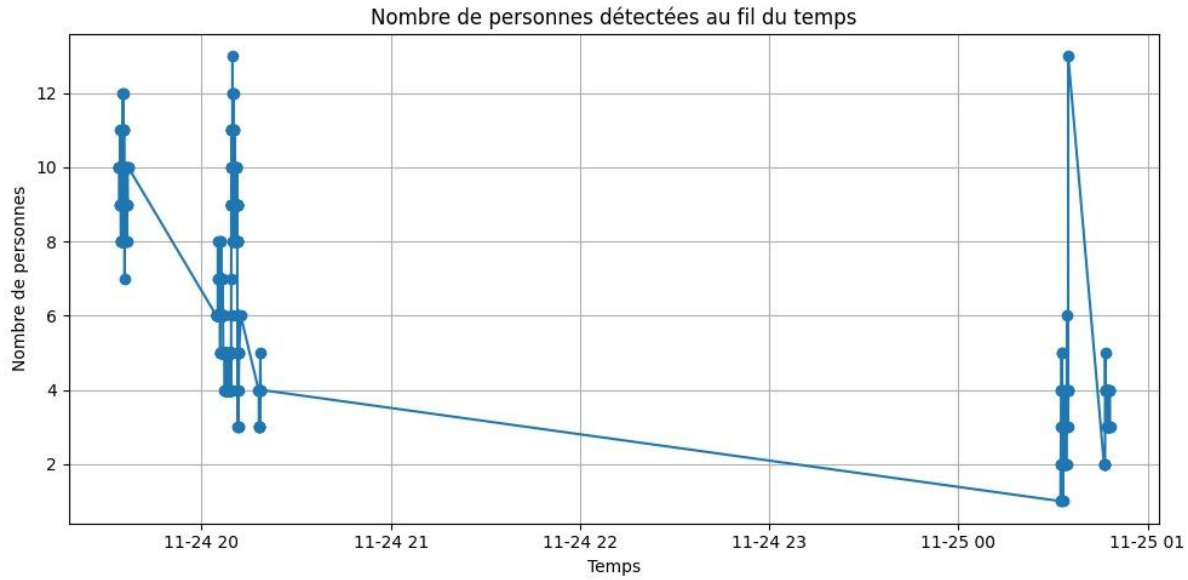
On observe le démarrage simultané des deux threads (cam1 et cam2) qui traitent chacun un flux vidéo distinct.

Dans la partie supérieure de l'image, on voit une image extraite du flux vidéo après traitement par le modèle YOLOv8, avec les personnes détectées entourées de cadres rouges.

Pour analyser ces détections, plusieurs graphiques ont été générés à l'aide de Matplotlib à partir des données enregistrées dans le fichier CSV :

- Courbe des détections par caméra : permet de comparer les détections issues de cam1 et cam2 séparément.
- Courbe globale : représente le nombre total de détections enregistrées par les deux caméras.
- Histogrammes d'activité par minute : permettent d'observer les périodes où l'activité est plus élevée.

Graphique global du nombre de personnes détectées :



Dans cette étape, nous avons regroupé toutes les données provenant des deux caméras, puis nous les avons fusionnées dans un seul fichier CSV pour pouvoir analyser l'activité globale.
Après avoir effectué le traitement des données, le graphique ci-dessous nous montre comment le nombre total de personnes détectées évolue au fil du temps.

Grâce à ce graphique, nous avons pu observer :

- les variations d'activité selon les horaires,
- les moments où on remarque un pic de détection,
- comment les deux caméras se complètent pour couvrir une zone plus large

Conclusion

Le système développé illustre une application pratique et performante de la détection de personnes en temps réel, couplée à la transmission d'images via Internet. Cette technologie se prête à de multiples domaines d'application :

- **Surveillance de sécurité** : elle peut être déployée dans des zones sensibles telles que les banques, les aéroports ou les entrepôts pour identifier et suivre les individus en temps réel.
- **Systèmes de contrôle d'accès** : intégrée à des bases de données, elle permet de vérifier automatiquement l'identité des personnes détectées.
- **Analyse de flux de clients** : dans les commerces ou centres commerciaux, elle offre des informations précieuses sur le comportement des clients, contribuant à l'optimisation des opérations et à l'amélioration de l'expérience utilisateur.

Ainsi, ce projet démontre non seulement la faisabilité technique de la détection et du suivi en temps réel, mais ouvre également la voie à des applications concrètes dans la sécurité, la gestion des accès et l'analyse comportementale.

Optimisations et Perspectives

Sécurisation des Données

Pour renforcer la sécurité des communications entre le client et le serveur, l’utilisation de AWS KMS (Key Management Service) et du chiffrement TLS est recommandée.

Reconnaissance Faciale

En combinant la détection de personnes avec des fonctionnalités de reconnaissance faciale via Amazon Rekognition, il serait possible d'identifier les individus détectés, améliorant ainsi la sécurité et l’efficacité du système.

Gestion Multicaméra et Extensibilité

Le système pourrait être étendu pour prendre en charge plusieurs flux vidéo simultanément, en utilisant AWS Elastic Load Balancer et EC2 Auto Scaling pour distribuer la charge et assurer la résilience.

Alertes et Notifications en Temps Réel

L’intégration de services comme Amazon SNS (Simple Notification Service) permettrait d'envoyer des notifications en temps réel par e-mail ou SMS en cas de détection spécifique.

Interface Web

Une interface web pourrait être développée sur AWS Amplify pour visualiser les détections en temps réel, consulter l’historique des données et configurer le système, permettant ainsi une gestion centralisée et intuitive.

Bibliographie et références

Voici les références des principales bibliothèques, outils et services utilisés dans ce projet.

→ **Ultralytics YOLOv8**

Ultralytics YOLOv8 for object detection in images and videos. Disponible sur : https://github.com/ultralytics/yolov8

→ **OpenCV**

OpenCV (Open Source Computer Vision Library). Disponible sur : https://opencv.org/

→ **FFmpeg**

FFmpeg, le logiciel de traitement vidéo. Disponible sur : https://ffmpeg.org/

→ **Amazon Rekognition**

Amazon Rekognition pour la détection d'objets dans les images. Disponible sur : https://aws.amazon.com/rekognition/

→ **Amazon Web Services (AWS)**

Documentation officielle d'AWS EC2 et S3 :
https://aws.amazon.com/fr/ec2/
https://aws.amazon.com/fr/s3/

→ **Boto3**

Boto3 documentation pour AWS : https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

→ **Pandas**

Pandas - Python Data Analysis Library. Disponible sur : https://pandas.pydata.org/

→ **Matplotlib**

Matplotlib pour la visualisation de données. Disponible sur : https://matplotlib.org/