**Implementation in real world scenario**

Implementing our person Re-Identification (ReID) and tracking pipeline in a **real-world commercial store** involves transforming your local script into a **scalable, cloud-integrated, and robust video analytics solution**. Below is a **step-by-step technical guide**, covering deployment architecture, Azure integration, hardware/software setup, performance considerations, and data handling.

**OVERVIEW**

We want to:

* Run person tracking and ReID on **30 GB+ CCTV video data**.
* Deploy it in a **real commercial store**, with CCTV integration.
* Store and access videos + outputs via **Azure cloud services**.
* Use it possibly for **customer analytics, movement tracking, or loss prevention**.

**STEP-BY-STEP TECHNICAL GUIDE**

**STEP 1: Assess the Environment**

**Store CCTV Setup**

* Confirm **camera specifications**: resolution, frame rate, codec (e.g., H.264).
* Check if **live stream** or **recorded video batches** are required for processing.
* Make sure cameras support **RTSP or HTTP video streaming**, or files can be dumped regularly to a server.

**Decide on Processing Mode**

* **Option A**: Batch processing – process 30GB of stored video.
* **Option B**: Real-time stream processing – continuous detection from RTSP stream.

You can support both with minor changes using OpenCV and GStreamer.

**STEP 2: Infrastructure Planning**

**Option 1: Edge + Cloud Hybrid**

* Use an **on-premise machine** (e.g., an NVIDIA Jetson or GPU desktop) to:
  + Pull video from CCTV or NVR.
  + Run YOLOv8 + DeepSORT + ReID.
  + Upload results + summaries to Azure Blob Storage or SQL DB.

**Option 2: Full Cloud Deployment (Azure)**

* Upload 30GB video files to **Azure Blob Storage**.
* Use **Azure VM with GPU** (e.g., NC-series) or **Azure Batch** to run your processing pipeline.
* Store outputs (CSV, annotated video) back in Blob Storage.

**STEP 3: Code Refactor for Production**

Refactor your Python code into a modular pipeline:

**video\_ingestion.py**

* Pulls videos from Azure Blob (using azure-storage-blob).
* Or listens to RTSP streams for live feeds (OpenCV or FFmpeg).

**detect\_and\_track.py**

* Your current person detection, tracking, and ReID pipeline.
* Add support for batch-wise processing to avoid memory overload.

**azure\_uploader.py**

* Uploads CSV results and annotated videos back to Azure.
* Implements retry, logging, error handling.

**Add Logging & Monitoring**

* Use logging, tqdm, and possibly **Azure Monitor** for logs and metrics.

**STEP 4: Azure Integration**

**Storage: Azure Blob Storage**

Use for:

* Input videos.
* Output results (CSV, annotated videos, metadata).

pip install azure-storage-blob

from azure.storage.blob import BlobServiceClient

# Connect

blob\_service\_client = BlobServiceClient.from\_connection\_string("AZURE\_CONN\_STRING")

container\_client = blob\_service\_client.get\_container\_client("video-data")

# Upload

with open("results.csv", "rb") as data:

container\_client.upload\_blob(name="results.csv", data=data, overwrite=True)

**Compute: Azure VM (NC-series) or Azure Batch**

* Create a VM with **NVIDIA GPU + CUDA drivers + Python env**.
* Install dependencies via requirements.txt.

**Tools**

* Use **Azure Functions** for event-driven triggers (e.g., when new video is uploaded).
* Use **Azure Batch** for large-scale processing of many videos in parallel.

**STEP 5: Performance Optimization**

**Hardware**

* If real-time is needed, use a system with **at least RTX 3060 or A5000 GPU**.
* For Azure: use **Standard\_NC6s\_v3 or NC12s\_v3 VMs**.

**Code Optimizations**

* Limit frame rate to 5–10 FPS if real-time not required.
* Skip small boxes, short tracks (<10 frames), etc.
* Use **batch inference** where possible.
* Cache embeddings or pre-filter low-quality detections.

**STEP 6: Output Management**

**Store per-video metadata**

* Save:
  + video\_name, frame\_idx, global\_id, face\_box, track\_coords
  + Store in **CSV** or write to **Azure SQL or CosmosDB** if analytics is needed.

**Annotated Videos**

* Compress output videos (ffmpeg) before uploading.
* Store in separate Blob container (annotated-videos/).

**STEP 7: Dashboard or Analytics (Optional)**

If the store wants to **view or analyze** data:

* Create a **Power BI dashboard** linked to Azure SQL DB or CSV files.
* Plot:
  + Number of unique people per hour.
  + Heatmaps of store zones (if camera zones are known).
  + Dwell time per customer.

**Tools & Technologies Recap**

| **Task** | **Tool** |
| --- | --- |
| Video Processing | OpenCV, YOLOv8, DeepSORT, Torchreid |
| Face Embedding | InsightFace |
| Cloud Storage | Azure Blob Storage |
| Compute | Azure GPU VM (NC series), Azure Batch |
| Logging | Python logging, Azure Monitor |
| Uploading | azure-storage-blob |
| Database (optional) | Azure SQL, CosmosDB |
| Dashboard (optional) | Power BI |

**Deployment Checklist**

* Azure account + Blob Storage container
* Azure VM with GPU (CUDA + Python)
* Modified Python pipeline for batch/live video
* Script to upload videos to Blob
* Script to process videos and save outputs
* Script to upload results back
* Optional: database + dashboard

**Security and Privacy**

* Ensure CCTV feeds are **encrypted**.
* Face data is biometric; ensure it is processed per **GDPR** or **local privacy laws**.
* Do not store raw face images unless explicitly allowed.