HTX - Person Re-Identification

Linking perspectives for a smarter, safer Singapore

August 2025

thoughtworks

Table of contents

Summary

This report outlines Thoughtworks' proposed approach for enabling large-scale Person Re-Identification (ReID) across Singapore's 50,000-camera Home Team network. The objective is to dramatically shorten investigation times, improve cross-agency intelligence sharing, and maintain operational accuracy while optimising compute and infrastructure costs.

We present a scalable ReID architecture built around three key processing stages:

1. **Detection and Tracking** – Efficiently detect and track people across tens of thousands of streams using GPU-accelerated models. Detector selection has the greatest impact on infrastructure cost; for example, PeopleNet-ResNet34 requires ~35 H100 GPUs, compared to over 435 for PeopleNet-ViT. Motion-based frame skipping can reduce effective processing from 10 FPS to 4 FPS per camera, cutting GPU demand by ~60%.
2. **ReID Feature Extraction** – Use a track-aware embedding policy to generate embeddings only at key moments (start, exit, periodic refresh). This reduces redundant inference and allows a small A10 GPU pool (6–10 GPUs) to handle embedding workloads with capacity for burst events.
3. **Multi-Camera Track Matching (MCTM)** – Link tracklets across non-overlapping cameras using spatio-temporal graphs, geometry-aware pruning, and probabilistic transition models. User interaction via Natural Language Query and visual crop selection seeds the matching process, with iterative approval/rejection loops improving accuracy and reducing false positives. At this scale, MCTM is CPU-bound for graph construction and solver logic, with GPUs reserved for large-scale similarity scoring or learning-based matchers.

This approach balances investigation speed, accuracy, and infrastructure efficiency, reducing GPU requirements by hundreds of units compared to naive implementations and enabling a future-ready ReID capability aligned with HTX's AI-first vision.

Person ReID

For HTX, person re-identification (ReID) is a critical capability in enabling Home Team agencies to track persons of interest across Singapore’s camera network. By linking detections from multiple cameras and locations, ReID supports faster investigations, coordinated responses, and enhanced situational awareness. The challenge lies in delivering this at a large scale, processing tens of thousands of streams in real time, while optimising for accuracy, compute efficiency.

State-of-the-art person ReID systems share a common set of core components, with variations largely in model architecture and processing pipeline. Across these pipelines, the person detector is typically the most computationally demanding stage. Detector choice has a significant impact on infrastructure requirements: for instance, PeopleNet-ViT requires roughly 13.6x more compute than PeopleNet-ResNet34, a difference that directly scales the number of GPUs needed to process high-volume video streams.

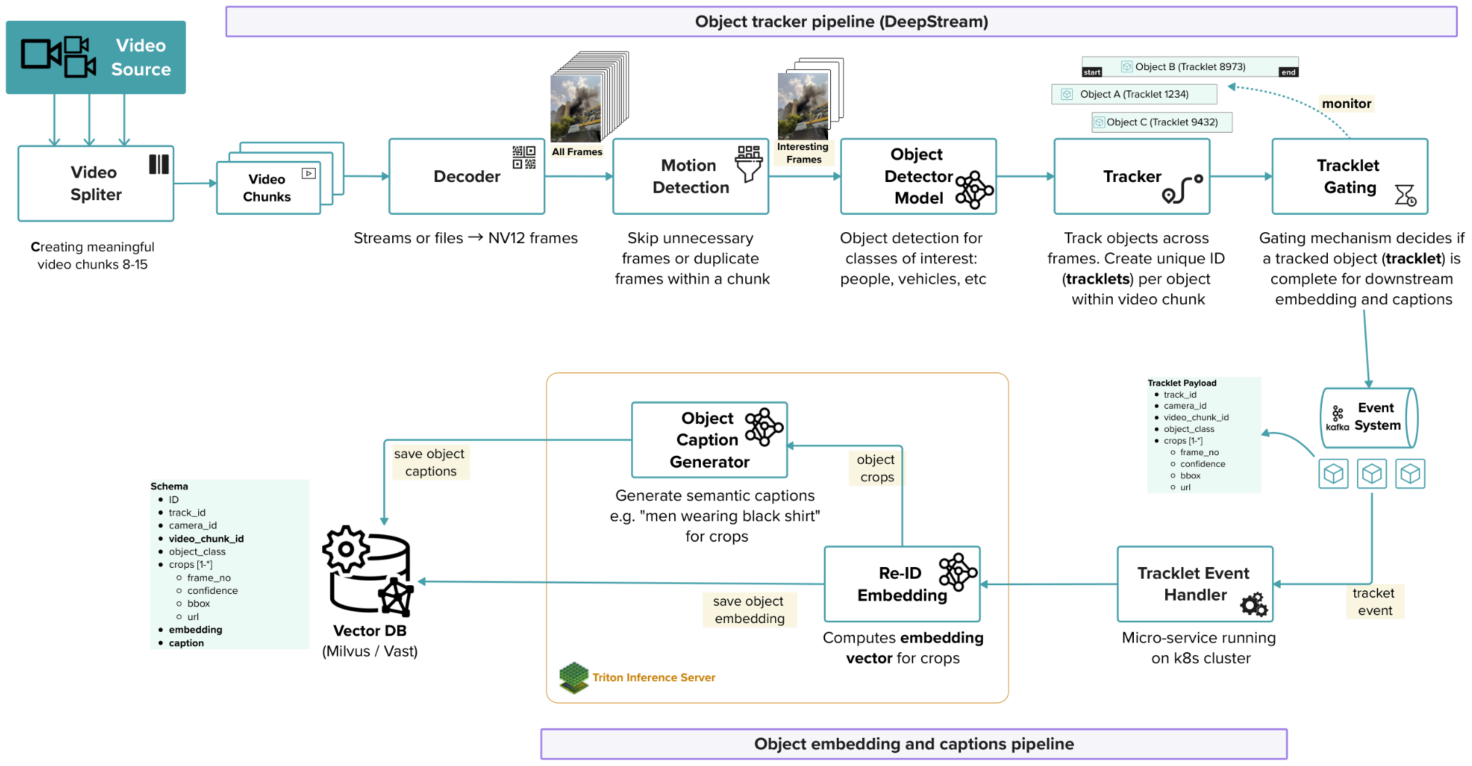
Common state-of-the-art ReID systems consist of the following components.

Component	Description	Examples
Person Detection	Localize people in frames using CNN or Transformer based models trained on large-scale person datasets.	CNN: ResNet, YOLOX Transformer: DINO-DETR, ViT, TransReID
Pose Estimation (optional)	Estimate key points to improve robustness under occlusion or for gait analysis. Pose estimation is becoming rarer in ReID benchmarks, but still valuable in heavy occlusion settings.	OpenPose, HRNet, Pose-Guided Feature Alignment
Feature Extraction	Generate discriminative embeddings for each person. Can use the embedding from the primary person detector or use another CNN or Transformer feature extractor to calculate another embedding.	CNN: ReIdentificationNet, PCB Transformer-based: TransReID, BOT
Single-Camera Tracking	Associate detections over time within a camera to produce continuous tracklets. Combines motion filters and appearance features.	DeepStream supports most of the common single camera trackers (NvDCF, DeepSORT). ByteTrack is used in many of the SOTA papers. There is a community-contributed project for DeepStream
Multi-Camera Matching	Link tracklets across cameras using embedding similarity and spatio-temporal constraints. Often uses camera locations & travel time models to filter tracklets and improve accuracy.	Graph matching Multi-Modal Multi-Camera Tracking dataset AI City Challenge multi-camera ReID winners,

Table 1. State-of-the-art person ReID systems share a common set of core components

Input Processing Pipeline

A large-scale person ReID system must efficiently balance thousands of concurrent camera streams, high detection accuracy, GPU resource constraints, and fast searchability of embeddings. At this scale, the GPU infrastructure is typically the single largest cost driver, so the input pipeline needs to be designed to minimise unnecessary processing while preserving critical events. Our approach combines motion-based frame filtering, multi-stream batching, and track-aware filtering to ensure that only meaningful person crops are passed to the ReID embedding and object caption stages. This not only reduces the compute footprint but also allows the ReID embedding service to scale independently from the detector, handling bursty loads during peak activity.



ReID input processing pipeline

The table below outlines the full input pipeline. In the following sections we will deep dive into key stages of the pipeline. This pipeline will work with real-time camera streams, or with video files.

Stage	Purpose	Based on 50k cameras
Camera Ingest	Handles network ingestion from IP cameras or files.	500k FPS
Decode	Hardware-accelerated video decode. Decodes H.264/H.265 into raw NV12 frames.	500k FPS
Motion Detection	Detects activity to trigger inference. Can be configured for ROI-based motion detection. If motion isn't detected the frame isn't sent to the Detector.	200k FPS
Stream batcher	Batch frames across cameras or files. Mandatory for multi-stream GPU efficiency. *Not shown in the diagram*	200k FPS
Detector	Full-frame object detection. Runs a detector model (PeopleNet, YOLO, PeopleNet-ViT) with TensorRT. **Outputs**: bounding boxes, class labels (Person), confidence, cropped image.	35-435 x H100 GPU, depending on model arch.
Tracker	Track objects on a single camera across frames. Uses a DeepStream Tracker (NvDCF, DeepSORT, or ByteTrack) algorithm to maintain track IDs over time. **Outputs**: tracking status (new, ongoing, lost), track confidence.	
Track-aware filtering	Decide which person crops to embed based on track status. Checks track time since last embed, IoU drift, occlusion, confidence then decides whether to send the image crop to ReID embedding.	

ReID Embedding	Computes embedding vectors for selected crops in a track.	11,250 embeddings / second
Object Caption Generator	Generate semantic captions e.g. "men wearing black shirt" for crops	11,250 captions / second

Multi-Camera Track Matching (MCTM)

Multi-Camera Track Matching (MCTM) links tracklets from different cameras by combining several complementary methods:

- Spatio-temporal graph association – representing tracklets as nodes, with edges encoding feasible links based on travel-time constraints and camera network topology.
- Geometry-aware pruning – applying calibration data, ground-plane homographies, and epipolar constraints to remove physically impossible matches.
- Probabilistic transition models – learning typical travel-time distributions between camera pairs to weight and filter candidate links.

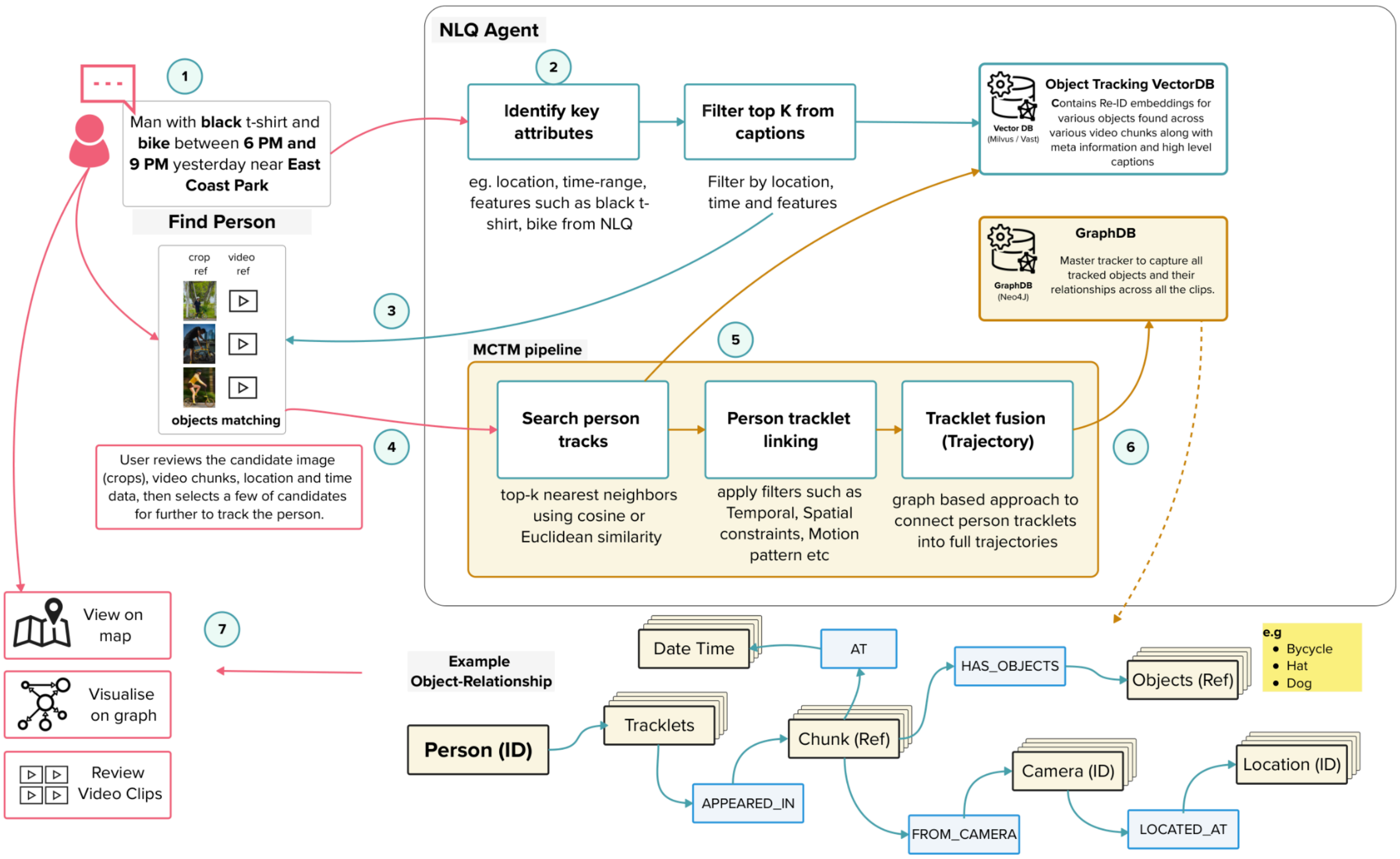

NLQ Agent
1
Man with black t-shirt and bike between 6 PM and 9 PM yesterday near East Coast Park
Find Person
crop ref
video ref
objects matching
User reviews the candidate image (crops), video chunks, location and time data, then selects a few of candidates for further to track the person.
2
Identify key attributes
eg. location, time-range, features such as black t-shirt, bike from NLQ
Filter top K from captions
Filter by location, time and features
Object Tracking VectorDB
Contains Re-ID embeddings for various objects found across various video chunks along with meta information and high level captions
Vector DB (Milvus / Vast)
GraphDB
Master tracker to capture all tracked objects and their relationships across all the clips.
GraphDB (Neo4J)
3
4
5
6
MCTM pipeline
Search person tracks
top-k nearest neighbors using cosine or Euclidean similarity
Person tracklet linking
apply filters such as Temporal, Spatial constraints, Motion pattern etc
Tracklet fusion (Trajectory)
graph based approach to connect person tracklets into full trajectories
7
View on map
Visualise on graph
Review Video Clips
Example Object-Relationship
Person (ID)
Tracklets
APPEARED_IN
Chunk (Ref)
AT
Date Time
HAS_OBJECTS
Objects (Ref)
e.g
• Bycycle
• Hat
• Dog
FROM_CAMERA
Camera (ID)
LOCATED_AT
Location (ID)

Figure 2: Officer using an Agent to find a person's track in a city using natural language query (NLQ) interface.

Consider the scenario where the office is initiating a person finding investigation using a Natural Language query to find a person's track in a city.

1. The query may specify time, location, and descriptive attributes to target relevant cameras and tracklets — for example "Man with black t-shirt and bike between 6 PM and 9 PM yesterday near East Coast Park"
2. The NLQ agent would Identify key attributes from the query such as **location**, **time-range** and **visual features** such as black t-shirt, bike.
3. Using these key attributes, the agent would query the Vector DB for track metadata, object crop images and reference video chunks and present it to the user.
4. The user reviews the candidate image (crops), video chunks, location and time data, then selects a few candidates to indicate further interest in tracking the respective person(s) tracks for a given time range and location.
5. This triggers the Multi-Camera Track Matching (MCTM) pipeline, which searches for likely matches across the camera network by combining visual similarity with spatio-temporal constraints
6. The identified tracklets are then fused into complete trajectories using a graph-based approach, where each tracklet is a node and connections are formed based on matching confidence. These fused trajectories are stored in a graph database (GraphDB), capturing complex relationships between observations across time and cameras
7. The results can then be presented to the user as a graph visualization, map-based person tracks, or raw video chunks for review. The user continues to interact with the NLQ Agent, providing feedback, applying filters, and refining the matches until the target is identified.

At the scale of 50,000 cameras, aggressive pruning through travel-time and geometry constraints keeps the candidate graph sparse, enabling near real-time operation. The graph construction and solver stages are dominated by branching logic and irregular memory access, making them primarily CPU-bound, while large-scale similarity scoring or learning-based association models can benefit from GPU acceleration.

Large-scale ReID

Our reference scenario is a 50,000-camera network used for person re-identification. At present, investigations are conducted manually, often taking several days to complete. The goal of the proposed solution is to dramatically shorten investigation times while maintaining acceptable accuracy.

For this scenario, we'll assume that edge computing isn't an option and camera streams or video clips are processed by a centralized GPU cluster of NVIDIA H100s. After the GPU cluster calculations we will explore network and storage requirements.

In large deployments, one of the most significant expenses is the GPU infrastructure itself. For a large scale network of cameras, even small changes in detector efficiency can translate into hundreds of additional GPUs and millions in hardware, energy, and operational costs. In this section, we focus our calculations on estimating and optimizing the number of GPUs required for real-time processing, comparing different detector options and exploring opportunities to reduce compute demand without sacrificing accuracy.

To size the GPU cluster, we start by estimating the total inference throughput required for person detection across the entire camera network, then divide by the sustained per-GPU throughput for a given detector. The baseline formula is:

$$Detector\ GPUs\ =\ \frac{N_{cameras} \times FPS_{processing}}{FPS_{per\ GPU\ (detector)}}$$

Where:

- $N_{cameras}$ = Total number of cameras in the network (50,000)
- $FPS_{processing}$ = Effective frames per second to be processed per camera
- $FPS_{per\ GPU\ (detector)}$ = Sustained detection throughput for the chosen model on an NVIDIA H100 using NVIDIA performance benchmarks.

For large camera networks, reducing the effective FPS of the entire network is an effective method to reduce compute and storage costs. Research on urban surveillance highlights that "almost all" video is unimportant most of the time (rare events/changes are sparse), highlighting an opportunity to skip most frames outside peak hotspots. For example, in a large city-based network, you can usually ignore ~60–90% of frames on average, and 90%+ for many quiet cameras, but busy hubs (CBD junctions, MRT platforms, malls) may only let you skip ~10–25%.

For our purposes, we can estimate the distribution of cameras across busy, medium and low activity regions. For these calculations, we'll assume that 40% of the cameras are located in busy areas, 30% in medium activity areas and 30% in low activity areas.

Camera Area Type	% of Frames in Network	Typical Skip Ratio (No Motion)	Contribution to Overall Skip %
Busy (CBD junctions, MRT platforms, malls)	40%	30% skipped	0.40 × 0.30 = 12%
Medium (secondary streets, mid-traffic venues)	30%	70% skipped	0.30 × 0.70 = 21%
Low (residential streets, office lobbies after hours, parking lots)	30%	90% skipped	0.30 × 0.90 = 27%

Table 2. Camera distribution across busy, medium and low activity areas, resulting in an overall skip rate of 60% (12% + 21% + 27%)

Based on the assumed camera distribution, we estimate that 60% of frames across the network can be skipped without impacting detection coverage. For cameras streaming at 10 FPS, this translates to an effective average processing rate of 4 FPS for each camera across the entire network.

Using this formula, we can estimate the size of a GPU cluster for multiple model architectures.

- Cameras: 50,000
- Actual camera FPS: 10 FPS
- Effective detector sampling rate across the entire network: 4 FPS per camera, although all video will be processed using the original 10 FPS
- Required detector throughput = 50,000 cameras x 4 FPS = 200,000 FPS for the network.

In the following table are the number of GPUs required for three detectors (PeopleNet-ResNet, PeopleNet-ViT and YOLOv9-t) using NVIDIA's H100 performance data. Calculations use end-to-end FPS figures (frame decode → pre processing → batching → inference → post processing).

Model Arch	Inference resolution	Precision	Tracker	H100 GPU (FPS)	Detector GPUs
PeopleNet-ResNet34	960×544	INT8		6831	30
PeopleNet-ResNet34	960×544	INT8	NvDCF (+16%)	5842	35
YOLOv9t	640×640	FP16		2688	75
YOLOv9t	640×640	FP16	NvDCF (+5%)	2541	79
PeopleNet- ViT	960×544	INT8		495	405
PeopleNet- ViT	960×544	INT8	NvDCF (+7%)	461	435

Table 3. Estimation of the number of GPUs required to support 50,000 cameras for multiple model architectures and with and without using an object tracker

Object Tracking

Performance results in Table 3 show that the cost of incorporating a tracker alongside the detector is minimal relative to the overall compute footprint. For example, pairing PeopleNet-ResNet34 with the NvDCF tracker increases GPU needs by roughly 16%, while YOLOv9-t rises by just 5%, and PeopleNet-ViT by 7%. In this case the NvDCF tracker was used, however, other trackers (DeepSORT or ByteTrack) could be substituted at this stage of the processing pipeline. Tracking at this stage of the pipeline is especially important because it provides a filter mechanism to reduce the load on the ReID Feature Extractor stage of the pipeline.

ReID Feature Extractor

Person ReID pipelines can use an additional feature extraction to calculate the embedding for a person crop, instead of using the embedding from the primary detector. The ReID feature extractor has a different workload profile from the detector stage. While the detector runs at a steady, predictable rate tied to the incoming frame stream, ReID feature extractor demands are inherently bursty. Embedding requests spike during high-activity periods, such as rush hour or incidents, and remain low during quieter periods. This variability requires the ReID stage to scale independently of the detector, matching GPU resources to real-time demand rather than fixed frame rates. Continuously sending every person crop for embedding is inefficient, as consecutive frames often provide minimal new spatial or temporal information. Selectively triggering ReID embedding based on contextual cues (e.g. pose, track history, or appearance) is therefore key to reducing waste while maintaining identification accuracy.

There are multiple mechanisms for reducing the number of person crops sent to the feature extractor stage, each aiming to prioritise only the most useful samples. Examples include pose-guided gating, which focuses on embeddings from favourable body orientations; history-aware embedding refinement, which uses past track information to decide when and how to update features, and appearance-based continuity methods, which cluster detections by visual similarity to maintain identity without re-embedding every frame. Any or all of these would be helpful to filter embedding inference.

For the purpose of estimating the GPU requirements for the embedding stage, let's design a simple 'track-aware' policy. The purpose of our policy is to trigger the feature extractor at the start and end of each track, and every T seconds only if something changes enough to justify a re-embed (e.g. IoU drift, pose change).

This policy allows us to estimate the average embedding requests per second per camera using the formula:

$$E_{cam} \approx \lambda\left(\frac{2}{D} + \frac{1}{T}\right)$$

Where:

- λ = average persons per processed frame.
- D = average track duration (seconds) a person stays visible.
- T = re-embed period (seconds), e.g., embed at start, every T seconds, and at exit.

If we assume the following defaults:

- λ = 0.33 (on average 3 out of 10 processed frames contain a person)
- D = 8 seconds (typical dwell through FOV)
- T = 2 seconds (re-embed cadence)

$$E_{cam} \approx 0.3\,(0.25 + 0.5) = 0.2475\ embeds\ /\ second$$

So for 50,000 cameras in the network, we can estimate that 11,250 embeddings are requested per second. We'll use the performance numbers for NVIDIA's TAO ReIdentificationNet model for our calculations assuming a crop resolution of 256×128. When we map the estimated throughput requirements of the camera networks we can determine the number of GPUs required. Based on NVIDIA's performance benchmarks A10 24 GB achieve a sustained throughput of ~5,000–8,000 embeds/s/GPU (batch 64–128). It is possible to use a H100 for ReID Feature Extraction, but it will be overkill.

Using a conservative 6,000 embeds/s per A10.

$$Feature\ Extraction\ GPUs = \frac{11{,}250}{6{,}000} \approx 1.9 \Rightarrow 2 \times A10s$$

The ReID feature extractor requirements look tiny because the filter policy is efficient (embed on start/exit + every 2 s). However, we know that requests will burst with events and crowding, so we must plan for capacity with buffers:

- Base pool: 2 x A10
- Burst factor 3-5x: start with 6-10 A10 GPUs to absorb peaks.

NVIDIA Roadmap

Person Re-Identification isn't supported in NVIDIA's VSS, however, many of the components required to build a state of the art Person ReID are available as open source tools and as DeepStream plugins. The Multi-Camera Tracking AI Workflow is a possible solution, however NVIDIA is working on developing a newer version of this tool to support large-scale city wide applications. A POC from the NVIDIA Metropolis team demonstrating this functionality is expected to be ready in Q4 2025, with the first iteration of this functionality ready in early 2026.

Person ReID Resources

For benchmarking multi-modal multi-camera tracking methods, MTMMC is a Large-Scale Real-World Multi-Modal Camera Tracking dataset that includes RGB and thermal camera data across diverse environments.

The AI City Challenge has become a flagship annual competition hosted at major CVPR and ICCV focusing on Multi-Camera 3D Perception. It is an excellent resource for state-of-the-art Re-ID paper and code. Each year, challenges participants to detect and consistently track individuals across synchronized indoor camera networks. The provided datasets combine real footage with realistic synthetic scenes generated through NVIDIA Omniverse.