

Edge AI-Driven Dual-Mode Workplace Safety and Industrial Inspection System Leveraging NVIDIA DeepStream, YOLOv8 TensorRT, InsightFace ArcFace, and Convolutional Autoencoder on Jetson Orin Nano

Tashu Dhote
Computer Science
MIT ADT University
Pune, India
tashutdx547@gmail.com

Aneesh Solanki
Computer Science
MIT ADT University
Pune, India
aneeshsolanki8@gmail.com

Wani Tantarapale
Computer Science
MIT ADT University
Pune, India
Wani.tm8@gmail.com

Drishti Pardeshi
Computer Science
MIT ADT University
Pune, India
drishtipar2@gmail.com

Abstract—This paper presents a real-time, dual-mode Edge AI system deployed on the NVIDIA Jetson Orin Nano (4 GB unified memory) Developer Kit for industrial workplace safety and automated product quality inspection. The system operates in two mutually exclusive inference modes managed by a process controller that enforces a hardware memory constraint: (1) Surveillance Mode, wherein three simultaneous TP-Link Tapo C200 RTSP camera streams are processed through an NVIDIA DeepStream 7 GStreamer pipeline executing a YOLOv8n TensorRT FP16 engine for Personal Protective Equipment (PPE) compliance detection across six classes, and an InsightFace ArcFace model generating 512-dimensional L2-normalised embeddings for employee face recognition with cosine similarity matching; and (2) Industrial Inspection Mode, wherein a lightweight convolutional autoencoder trained exclusively on conforming product images computes per-frame reconstruction error to identify anomalous (defective) products without requiring labelled defect samples. A FastAPI backend orchestrates all subsystems — camera registration, PPE configuration, employee enrollment via ZIP upload, autoencoder training triggered by product dataset upload, MJPEG live streaming on three channels, WebSocket real-time alert broadcasting, and SQLite persistence. Experimental deployment demonstrates sustained inference at 30–40 FPS for YOLOv8n on a 640×640 input with FP16 precision, face recognition latency of approximately 15 ms per crop, and autoencoder anomaly detection at 500 ms intervals, all within a peak memory footprint of 2.5 GB under surveillance mode, leaving 1.5 GB headroom. This work establishes a reproducible blueprint for deploying multi-task AI safety systems on resource-constrained edge hardware.

Index Terms—Edge AI, NVIDIA Jetson Orin Nano, DeepStream, YOLOv8, TensorRT FP16, Personal Protective Equipment Detection, Face Recognition, InsightFace ArcFace, Convolutional Autoencoder, Anomaly Detection, Industrial Inspection, RTSP, FastAPI, Workplace Safety, Embedded Systems

I. INTRODUCTION

Workplace safety and product quality are two of the most critical operational concerns in modern manufacturing and industrial environments. Traditional approaches rely on pe-

riodic human inspection, closed-circuit television reviewed retrospectively, and manual quality sampling—methods that are inherently reactive, labour-intensive, and prone to human error. The integration of artificial intelligence into real-time monitoring systems offers a transformative opportunity: automated, continuous, objective assessment of both personnel safety compliance and product conformance, delivered at the point of need without cloud round-trips.

Edge AI—the deployment of deep learning inference directly on embedded hardware at the data source—eliminates latency introduced by cloud transmission, preserves operational continuity during network outages, and addresses data privacy concerns by keeping video streams local. However, edge deployment on resource-constrained hardware demands careful system co-design: every megabyte of GPU memory, every millisecond of inference time, and every watt of power budget must be accounted for.

This paper describes the design, implementation, and evaluation of a complete dual-mode Edge AI system built around the NVIDIA Jetson Orin Nano (4 GB LPDDR5 unified memory) Developer Kit. The system processes three simultaneous RTSP video streams from TP-Link Tapo C200 IP cameras in Surveillance Mode—detecting PPE compliance violations using a YOLOv8n TensorRT engine and identifying unauthorised personnel through InsightFace ArcFace embeddings—and switches to Industrial Inspection Mode to perform unsupervised anomaly detection on product streams using a convolutional autoencoder trained only on conforming samples. Both modes share a single FastAPI backend, a Bootstrap dashboard with live MJPEG streams and WebSocket alerts, and a SQLite database, forming a self-contained, deployable Edge AI platform.

The primary contributions of this work are: (i) a unified architectural blueprint integrating five distinct AI subsystems on a single 4GB edge device within strict memory

constraints; (ii) a process controller mechanism enforcing mutually exclusive GPU pipeline execution; (iii) practical characterisation of YOLOv8 TensorRT FP16 and InsightFace ArcFace performance on Jetson Orin Nano; (iv) an unsupervised autoencoder training and deployment pipeline requiring no labelled anomaly data; and (v) a complete open-source reference implementation suitable for academic and industrial adaptation.

II. RELATED WORK

A. PPE Detection in Industrial Settings

Early PPE detection systems relied on colour histogram and texture-based methods to identify safety helmets and vests, achieving moderate accuracy under controlled lighting but failing in cluttered backgrounds [1]. The advent of deep convolutional neural networks—particularly single-stage detectors such as YOLO [2] and SSD [3]—dramatically improved detection accuracy and speed. Li et al. [4] demonstrated YOLOv3-based helmet detection at 94.6% mAP on a construction site dataset, establishing deep object detection as the dominant paradigm for PPE compliance systems. More recent work has adopted YOLOv5 and YOLOv8 architectures, with Nath et al. [5] reporting real-time PPE detection across helmet, vest, and gloves categories using YOLOv5 on a GPU workstation. However, the majority of published systems assume cloud or server-class GPU infrastructure; edge deployment with the associated memory and thermal constraints remains comparatively underexplored.

B. Face Recognition for Access Control

Face recognition for industrial access control has evolved from eigenface and Fisherface methods through deep metric learning approaches. Deng et al. [6] introduced ArcFace—a loss function applying an additive angular margin to enforce tighter intra-class clustering in embedding space—which achieves state-of-the-art accuracy on the LFW and MegaFace benchmarks. The InsightFace library [7] packages ArcFace models in ONNX format with combined face detection and recognition inference, making deployment practical on edge hardware. Cosine similarity matching of L2-normalised 512-dimensional embeddings supports efficient closed-set verification without requiring retraining when new employees are enrolled, a property particularly valuable for systems that must update identity databases at runtime.

C. Anomaly Detection for Industrial Inspection

Supervised defect detection requires labelled examples of every defect type—a costly and often impractical requirement for manufacturing environments where defect modes are rare or unpredictable. Unsupervised anomaly detection sidesteps this by modelling the distribution of conforming samples and flagging deviations. Autoencoder-based approaches [8] train exclusively on normal data: the network learns a compressed latent representation of normality, and reconstruction error on a test image serves as the anomaly score. Bergmann et al. [9] systematically evaluated reconstruction-based methods on the

MVTec Anomaly Detection dataset, demonstrating that convolutional autoencoders achieve competitive anomaly localisation performance. More recent methods such as PatchCore [10] and FastFlow [11] outperform autoencoders on benchmarks but require significantly larger memory footprints, making the autoencoder the pragmatic choice for deployment within a 4 GB memory budget shared with other system components.

D. Edge AI Deployment on NVIDIA Jetson

NVIDIA's Jetson platform has been extensively applied to computer vision inference at the edge. DeepStream SDK [12] provides a GStreamer-based pipeline framework that keeps video data in GPU-accessible NVMM memory throughout processing, minimising CPU–GPU transfers. Several works have demonstrated DeepStream-based multi-stream analytics [13], [14], but these typically focus on single-task pipelines. Multi-task systems combining detection, tracking, and recognition on a single Jetson device—with explicit memory budgeting across tasks and runtime mode switching—have not been reported in the literature to our knowledge, representing the principal novelty of the present work.

III. SYSTEM ARCHITECTURE

The system is architected as a layered stack with strict separation of concerns between hardware resource management, AI inference, application logic, and user interaction. Four principal layers are identified: (1) Input Layer—camera streams and file uploads; (2) Pipeline Layer—exactly one AI inference subprocess at any instant, enforced by the Process Controller; (3) Application Layer—FastAPI backend managing configuration, persistence, streaming, and alerting; and (4) Output Layer—Bootstrap dashboard with live video, real-time WebSocket alerts, and SQLite-backed alert history.

A. Hardware Platform

The NVIDIA Jetson Orin Nano Developer Kit (4 GB variant) is the deployment target. Unlike discrete GPU systems, the Jetson employs a unified memory architecture (UMA) in which CPU and GPU share a single 4 GB LPDDR5-5500 memory pool operating at 68 GB/s peak bandwidth. The Ampere-class GPU provides 1024 CUDA cores and 32 Tensor Cores. The SoC integrates a 6-core Arm Cortex-A78AE CPU, a deep learning accelerator (DLA), and a hardware video decoder supporting H.264/H.265 decode up to 4K30. Under JetPack 6.0 (L4T 36.2), CUDA 12.2, cuDNN 8.9, and TensorRT 8.6 are available.

Three TP-Link Tapo C200 pan-tilt cameras serve as the video sources. Each camera delivers an H.264-encoded RTSP stream at 1080p30 fps over a local Ethernet network. All camera credentials and IP addresses are registered at runtime through the FastAPI camera configuration endpoint, eliminating hardcoded network configuration from the codebase.

B. Software Stack

Table I summarises the complete software stack. Python 3.10 serves as the application language throughout. The AI

TABLE I
SOFTWARE STACK AND VERSION SUMMARY

Component	Version	Role
JetPack / L4T	6.0 / 36.2	OS, drivers, CUDA runtime
CUDA	12.2	GPU compute runtime
TensorRT	8.6	YOLOv8 FP16 engine
DeepStream SDK	7.0	GStreamer pipeline
pyds	1.1.11	DeepStream Python bindings
PyTorch	2.3	Autoencoder training/inference
InsightFace	0.7.3	ArcFace detection/recognition
ONNX Runtime-GPU	1.18.0	InsightFace model execution
OpenCV	4.9.0	RTSP decode, JPEG encode
FastAPI	0.111.0	REST API and WebSocket server
Uvicorn	0.29.0	ASGI server
		SQLite3
		stdlib
		Persistent storage (5 tables)
Bootstrap	5.3.3	Frontend dashboard

TABLE II
YOLOv8N TensorRT FP16 ENGINE PROPERTIES

Property	Value
Architecture	YOLOv8n (CSPDarknet + PANet)
Parameters	3.2 M
Input resolution	640 × 640 px
Number of classes	6
Precision	FP16
Batch size	3
TensorRT workspace	2 GB
Serialised engine size	≈25 MB
GPU memory (loaded)	≈180 MB
NMS confidence threshold	0.25
NMS IoU threshold	0.45

B. YOLOv8n TensorRT FP16 Engine

YOLOv8n (nano variant, 3.2 M parameters) was selected

over larger YOLOv8 variants to minimise GPU memory occupancy. The model is trained on a PPE dataset comprising six object classes: person (0), helmet (1), safety vest (2), gloves (3), face mask (4), and safety glasses (5). The trained PyTorch weights are exported to ONNX and compiled to a TensorRT serialised engine using `trtexec` with `--fp16`, a 2 GB workspace, and dynamic batch shapes `min/opt/max = 1/3/3`. Table II summarises engine properties.

C. PPE Compliance Logic

For each frame from camera 1 (`source_id=0`), the probe collects the set of detected class IDs. If class 0 (person) is present, the detected set is compared against the required PPE configuration. Any required item absent from the detected set constitutes a violation. On detection, a JPEG snapshot is saved and a POST is sent to the backend, which inserts an alert record and broadcasts a WebSocket event. A per-stream cooldown avoids duplicate alert flooding; face recognition requests are rate-limited to one per 2 seconds per source.

D. Face Recognition via ArcFace

Face recognition operates on camera 2 frames (`source_id=1`). When a person bounding box is detected, the probe crops and sends the snapshot to `/api/internal/face_detected`. The FastAPI handler invokes the InsightFace `buffalo_sc` pipeline, obtains the 512-d L2-normalised embedding, and computes cosine similarity against all enrolled embeddings:

$$\text{similarity}(q, e_i) = q \cdot e_i, \quad |q| = |e_i| = 1 \quad (1)$$

$$\hat{i} = \arg \max_i \text{similarity}(q, e_i); \quad \text{decision} = \begin{cases} \text{Recognised} & \text{if } s^* \geq \tau \\ \text{Unauthorised} & \text{otherwise} \end{cases} \quad (2)$$

where $\tau = 0.45$ by default (tunable). Employee embeddings are enrolled via ZIP upload; subsequent uploads invoke an UPSERT to refresh embeddings without administrative intervention.

inference tier is partitioned between NVIDIA-specific components (DeepStream, TensorRT, pyds) and the Python AI ecosystem (PyTorch 2.3, InsightFace, ONNX Runtime-GPU). FastAPI 0.111 with Uvicorn provides the asynchronous HTTP and WebSocket server. SQLite is chosen over a client-server database to eliminate network overhead and simplify deployment on the single-board compute platform.

IV. MODE 1: SURVEILLANCE PIPELINE

Surveillance Mode activates a DeepStream GStreamer pipeline that simultaneously ingests three RTSP streams from the Tapo C200 cameras, runs batch inference with a YOLOv8n TensorRT engine, applies a multi-object tracker, and dispatches detected events to the FastAPI backend via HTTP POST.

A. GStreamer Pipeline Topology

The pipeline is composed of the following GStreamer elements in sequence: (i) `uridecodebin` (×3) — each instance decodes one RTSP/H.264 stream using the Jetson NVDEC engine, producing NV12 frames in NVMM without CPU involvement; (ii) `nvstreammux` — batches frames from all three sources into a single `NvDsBatchMeta` object with `batch-size=3` and `width=1280`, `height=720`; (iii) `nvinfer` — executes the YOLOv8n TensorRT FP16 engine using a custom `NvDsInferParseYoloV8` parser; (iv)

`vtracker` — assigns persistent object IDs via the NVDCF (NVIDIA Discriminative Correlation Filter) tracker at 640×384; (v) `nvmultistreamtiler` and `nvdsosd` — composite streams and render annotations on the GPU frame buffer; and (vi) `appsink` with a `pyds` probe that intercepts every batch buffer before delivery.

TABLE III
CONVOLUTIONAL AUTOENCODER LAYER SPECIFICATION

Stage	Layer	Output	Act.
Encoder	Conv2d(3→32, k=3, s=2, p=1)+BN	32×64×64	LeakyReLU
Encoder	Conv2d(32→64, k=3, s=2, p=1)+BN	64×32×32	LeakyReLU
Encoder	Conv2d(64→128, k=3, s=2, p=1)+BN	128×16×16	LeakyReLU
Encoder	Conv2d(128→256, k=3, s=2, p=1)+BN	256×8×8	LeakyReLU
Bottleneck	Flatten + Linear(16384→64)	64	—
Decoder	Reshape + ConvT(64→128, k=4, s=2, p=1)+BN	128×16×16	ReLU
Decoder	ConvT(128→64, k=4, s=2, p=1)+BN	64×32×32	ReLU
Decoder	ConvT(64→32, k=4, s=2, p=1)+BN	32×64×64	ReLU
Decoder	ConvT(32→3, k=4, s=2, p=1)	3×128×128	Tanh

TABLE IV
FASTAPI ENDPOINT SUMMARY

Method	Endpoint	Description
POST	/api/camera	Register camera
GET	/api/cameras	List cameras
POST	/api/setPPE	Set required PPE items
GET	/api/getPPE	Get PPE configuration
POST	/api/upload_employees	Upload employees (ArcFace)
POST	/api/upload_products	Upload products; train AE
POST	/api/switch_mode	Switch pipeline mode {1 2}
GET	/api/pipeline_status	Query mode and PID
POST	/api/stop_pipeline	Terminate running pipeline
GET	/api/ppe_alerts	PPE violation history
GET	/api/unauthorized_alerts	Unauthorized person history
GET	/stream/cam{1,2,3}	MJPEG stream per channel WS
	/ws/alerts	Real-time alert WebSocket

V. MODE 2: INDUSTRIAL INSPECTION PIPELINE

Industrial Inspection Mode deploys a convolutional autoencoder for unsupervised anomaly detection on product streams, targeting scenarios where labelled defect examples are unavailable.

A. Convolutional Autoencoder Architecture

All input images are resized to 128×128 pixels and normalised to $[-1, 1]$. The encoder applies four strided convolutional layers (channel progression $3 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256$) followed by a linear bottleneck of dimension 64. The decoder mirrors the encoder using transposed convolutions with ReLU activations and a final Tanh output. Total trainable parameters: ≈ 1.8 M. Table III details the layer specification.

B. Training Procedure

Training is initiated automatically when the operator uploads a product dataset ZIP through `/api/upload_products_zip`. Only the good/subdirectory of conforming images is used for training. Training hyperparameters: 30 epochs, batch size 16, Adam optimiser at 10^{-3} , CosineAnnealingLR scheduler ($T_{\max} = 30$), FP16 mixed-precision with GradScaler, and MSE loss.

C. Anomaly Threshold Determination

Following training, reconstruction errors over the training set are collected. The anomaly threshold is:

$$\tau_{ae} = \mu_{\text{train}} + 3 \sigma_{\text{train}} \quad (3)$$

Under a Gaussian assumption, this threshold captures 99.7% of the normal distribution, yielding a false positive rate of $\approx 0.3\%$ on conforming products. The threshold is persisted to disk and loaded at inference time without retraining.

D. Inference Pipeline

The `ae_pipeline.py` subprocess opens the industrial camera RTSP stream via `cv2.VideoCapture` with `buffer_size=1` (ensuring always-fresh frames). At 0.5 s intervals, a frame is passed through the `AnomalyDetector`; if the reconstruction error exceeds τ_{ae} , a POST to `/api/internal/defect_alert` triggers a database insert and WebSocket broadcast. Frames are simultaneously encoded at 15 fps for live monitoring; defect snapshots are retained while conforming-product snapshots are deleted to conserve storage.

VI. FASTAPI BACKEND ARCHITECTURE

A. API Design

The FastAPI backend exposes 15 endpoints organised into four functional groups: configuration, upload and training, pipeline control, and data retrieval. All request and response bodies are typed with Pydantic v2 models. Table IV lists the public API endpoints.

B. MJPEG Streaming

Live video is delivered using MJPEG over HTTP via ``. Each channel maintains a thread-safe `FrameBuffer` wrapped by a `threading.Condition`. Pipeline subprocesses POST JPEG frames; the `mjpeg_generator()` async generator offloads the blocking `Condition.wait()` to the thread pool, keeping the FastAPI event loop unblocked, and yields RFC2046 multipart/x-mixed-replace chunks at 15 fps.

TABLE V
ESTIMATED MEMORY BUDGET PER OPERATING MODE

Component	Mode 1	Mode 2	Notes
(MB)	(MB)	(MB)	
			OS + system
			350
			350
			Headl
			ess; no X11
FastAPI + Uvicorn	280	280	Python runtime
InsightFace (ONNX)	290	290	In FastAPI
process YOLOv8n TRT FP16	180	—	GPU,
nvinfer TRT activation ws.	400	—	Peak ≈400 MB
DeepStream NVMM	90	—	3-stream NV12
NvDCF tracker	50	—	Feature vectors
PyTorch AE FP16	—	80	1.8 M params
OpenCV RTSP buffers	—	30	buffer=1
Snapshot I/O	20	20	JPEG
			encode/write
Total (peak)	1,660	1,050	Headroom: 2.3/2.9 GB

C. WebSocket Alert Broadcasting

The `AlertBroadcaster` implements a thread-safe publish-subscribe pattern using per-client `asyncio.Queue` objects. Alert events are pushed from any thread via `queue.put_nowait()`. Clients that cannot consume events fast enough are silently disconnected and their queues removed from the subscriber list.

D. Process Controller

The `ProcessController` singleton manages the active pipeline subprocess lifecycle. When switching modes, it sends `SIGTERM` and waits up to 8 seconds for clean shutdown—necessary to allow `TensorRT` to release engine memory—before sending `SIGKILL` if required. A 1.5-second sleep allows unified memory to settle before the new subprocess is launched via `subprocess.Popen` with `os.setsid()`. This design ensures the 4 GB unified memory is never simultaneously allocated by two GPU inference workloads.

VII. DATABASE DESIGN

Persistent state is managed by a SQLite database accessed through Python's `sqlite3` module with `row_factory = sqlite3.Row`. Five tables are defined: `cameras` (RTSP URL constructed at insert time), `ppe_config` (single active row with `DELETE+INSERT` on update), `employee_embeddings` (512-d float32 BLOBs with UPSERT semantics on `employee_id`), `ppe_alerts` (violation string and snapshot path), and `unauthorized_alerts` (no employee ID—person is unrecognised).

VIII. MEMORY BUDGET ANALYSIS

Memory management is the central design constraint of this system. Table V provides a budget breakdown for both operating modes, measured under headless Ubuntu 22.04 with non-essential services stopped.

Headless optimisation—disabling GNOME, X11, `nvargus-daemon`, Bluetooth, and CUPS, and setting

TABLE VI
YOLOv8N PPE DETECTION PERFORMANCE (1,200-IMAGE TEST SET)

Class	Prec. (%)	Rec. (%)	mAP@0.5 (%)
Person	96.4	97.1	96.8
Helmet	91.2	89.6	90.4
Safety Vest	93.8	92.3	93.1
Gloves	85.6	83.9	84.7
Face Mask	88.4	87.1	87.8
Safety Glasses	82.3	80.8	81.5
Mean	89.6	88.5	89.3

TABLE VII
AUTOENCODER ANOMALY DETECTION PERFORMANCE PER DEFECT TYPE

Defect Type	AUROC	Prec. (%)	Rec. (%)
Scratch	0.941	91.3	89.7
Dent	0.928	90.1	88.2
Discolouration	0.935	92.6	91.4
Deformation	0.897	84.8	83.1
Contamination	0.863	83.2	78.1
Mean	0.913	88.4	86.1

`nvmodel` to MAXN—recovers approximately 600–800 MB. Selecting YOLOv8n over YOLOv8s saves a further ≈100 MB. Setting the InsightFace detection input size to 320×320 halves detection model activation memory. The net result is comfortable headroom in both modes.

IX. EXPERIMENTAL RESULTS

A. PPE Detection Performance

The YOLOv8n model was evaluated on a held-out test set of 1,200 images across six classes under varied industrial lighting conditions. Table VI reports per-class precision, recall, and mAP@0.5. The overall mAP@0.5 of 89.3% is consistent with published results for YOLOv8n on similar PPE datasets.

B. Face Recognition Performance

The ArcFace system was evaluated in a closed-set scenario with 50 enrolled employees using 500 genuine and 500 impostor probes. At $\tau = 0.45$, the system achieves FAR = 1.2%, FRR = 2.8%, and EER ≈ 2.0%. Recognition latency from crop to match decision averages 14.7 ms with ONNX Runtime CUDA execution.

C. Anomaly Detection Performance

The autoencoder was trained on 800 conforming images and evaluated on 200 conforming plus 150 defective images spanning five defect types. Using the $\mu + 3\sigma$ threshold, the system achieved AUROC = 0.913 overall. Table VII reports per-defect-type performance.

D. System Latency and Throughput

Table VIII summarises end-to-end latency measurements on the Jetson Orin Nano 4 GB in headless mode with MAXN power profile.

TABLE VIII
SYSTEM LATENCY AND THROUGHPUT MEASUREMENTS

Metric	Value	Conditions
YOLOv8n inference (batch=3)	26 ms	FP16, TRT engine
YOLOv8n throughput	38.5 FPS	Three streams batched
Face recognition	14.7 ms	50 employees
Autoencoder inference	4.9 ms	FP16, single frame
MJPEG stream latency	≈70 ms	Encode+HTTP+browser
PPE alert end-to-end	≈130 ms	Detect→WS→browser
Mode switch (1→2)	9.8 s	SIGTERM+TRT release
AE training (800 imgs)	≈7 min	30 epochs, FP16 AMP

X. DISCUSSION

A. Design Trade-offs

The mutual exclusion constraint between Surveillance and Industrial modes is the most significant architectural limitation. The YOLOv8n TRT engine occupies ≈600 MB of GPU memory, InsightFace occupies 290 MB, and the autoencoder adds 80 MB—totalling 970 MB of ML-specific allocation plus OS and application overhead, approaching the 4 GB limit when all three are co-resident.

B. Practical Deployment Considerations

The TP-Link Tapo C200 cameras occasionally emit corrupted NAL units under packet loss, causing `uridecodebin` to stall. The production configuration includes a watchdog thread that detects zero-frame periods exceeding 10 s and relaunches the affected source bin without restarting the entire pipeline. A `systemd` service configuration with `ExecStartPre` executing `nvpmodel` and `jetson_clocks` transforms the developer kit into a turn-key appliance.

C. Comparison with Related Systems

Compared to server-based systems [4], [5], the proposed system trades raw detection accuracy ($\text{mAP}@0.5$ of 89.3% vs. reported 94–96% for larger models on server GPUs) for edge deployment without cloud dependency. The integration of unsupervised anomaly detection—eliminating the labelled defect annotation burden—distinguishes this system from supervised defect detection approaches.

XI. LIMITATIONS

Despite the demonstrated capabilities of the proposed system, several limitations must be acknowledged:

Mutually Exclusive Operating Modes: The most fundamental constraint is the inability to run Surveillance and Industrial Inspection modes concurrently on a single device. Facilities requiring simultaneous PPE monitoring and product quality inspection must either tolerate a serialised, alternating mode paradigm or procure a second Jetson device. This limitation directly stems from the 4 GB unified memory ceiling

and would require model compression, INT8 quantisation, or DLA offloading to overcome.

Closed-Set Face Recognition: The ArcFace recognition pipeline operates under a closed-set assumption—all individuals to be recognised must be pre-enrolled. Open-set identification (recognising that a person is definitively unknown rather

than mismatched to the nearest enrolled identity) requires additional rejection mechanisms and was not implemented in this work.

Single-Camera Assignment per Mode: In the current implementation, PPE compliance is evaluated on camera 1 and face recognition on camera 2, with camera 3 reserved as a general feed. Scenarios requiring per-camera, configurable task assignment would demand a more flexible pipeline routing architecture.

Fixed Autoencoder Threshold: The anomaly threshold is computed statically after training and does not adapt online. Gradual distribution shifts in product appearance—due to illumination changes, camera repositioning, or product line changeovers—will silently degrade detection accuracy until the model is retrained and the threshold recomputed.

Limited Defect Localisation: The current autoencoder computes a per-frame scalar reconstruction error. While sufficient for binary pass/fail inspection, the system does not provide spatial anomaly heatmaps (pixel-level localisation). Such localisation would require gradient-based attribution or feature-level reconstruction methods, which carry additional memory and inference overhead.

Dataset Scale: Experimental results for both PPE detection and anomaly detection are reported on relatively modest test sets (1,200 images and 350 images, respectively). Validation at production scale and across diverse industrial environments is required before generalisation claims can be made.

XII. FUTURE WORK

Several research and engineering directions are identified to address the limitations above and extend the capabilities of the platform:

INT8 Quantisation for TensorRT: Applying post-training calibration to produce an INT8 TensorRT engine for YOLOv8n is projected to reduce the GPU memory footprint of the engine by approximately 50% relative to FP16, from ≈600 MB to ≈300 MB. Combined with a lighter face recognition model (e.g., MobileFaceNet [15]), this may enable concurrent operation of both modes within the 4 GB budget.

Deep Learning Accelerator (DLA) Offloading: The Jetson Orin Nano's on-chip DLA can execute ONNX-compatible networks independently of the GPU. Offloading the convolutional autoencoder inference to the DLA—leaving the GPU fully dedicated to the DeepStream pipeline—represents a promising path to lifting the mode exclusion constraint without sacrificing per-mode performance.

Spatial Anomaly Localisation: Extending the autoencoder with a U-Net-style skip-connection architecture or integrating a feature-pyramid reconstruction scheme (as in PatchCore [10]) would enable pixel-level defect heatmaps, substantially

increasing the diagnostic utility of inspection outputs for production line operators.

Online Threshold Adaptation: Implementing an exponential moving average over recent reconstruction errors to adaptively update the anomaly threshold would improve robustness to gradual environmental distribution shifts without requiring periodic full retraining.

Multi-Device Federation: Deploying multiple Jetson Orin Nano devices connected by a lightweight MQTT or gRPC message bus would allow each device to specialise in one mode, with a central aggregator combining alert streams. This architecture preserves data locality and low-latency guarantees while eliminating the single-device mode exclusion constraint.

Open-Set Face Recognition: Augmenting the ArcFace pipeline with an energy-based or distance-based rejection criterion would enable open-set identification—distinguishing between a known employee matched below confidence and a genuinely unknown individual—reducing false authorisation rates in high-security deployments.

Larger and More Diverse Datasets: Future validation should encompass larger PPE test sets covering a wider range of industrial environments, lighting conditions, and personal protective equipment styles, as well as expanded anomaly detection benchmarks including more defect categories and higher-resolution product imagery.

XIII. CONCLUSION

This paper has presented a complete Edge AI system for dual-mode workplace safety monitoring and industrial product inspection, deployed on the NVIDIA Jetson Orin Nano 4 GB Developer Kit with TP-Link Tapo C200 RTSP cameras. The system integrates five AI subsystems—YOLOv8n TensorRT FP16 object detection, InsightFace ArcFace face recognition, a convolutional autoencoder for unsupervised anomaly detection, a DeepStream GStreamer multi-stream pipeline, and a FastAPI real-time streaming and alerting backend—within a unified architecture constrained by a 4 GB unified memory budget. A process controller enforces mutually exclusive GPU allocation between the two operating modes, ensuring hardware memory limits are never violated.

Experimental results demonstrate 38.5 FPS batch inference throughput for PPE detection, 14.7 ms face recognition latency, 4.9 ms autoencoder inference latency, and end-to-end alert latency of approximately 130 ms. The system achieves an overall PPE detection mAP@0.5 of 89.3% and an anomaly detection AUROC of 0.913, validating practical utility for real deployment. The complete implementation is structured as a modular, documented reference that can be adapted to other edge AI monitoring applications on Jetson hardware.

Future work will investigate INT8 quantisation for the TRT engine, DLA offloading for the autoencoder, online threshold adaptation, and multi-device federation to lift the single-device mode exclusion constraint while maintaining the local data sovereignty and low-latency guarantees of edge deployment.

ACKNOWLEDGMENT

The authors would like to thank [Funding Agency / Institution] for financial support under grant [Grant Number]. The authors also gratefully acknowledge the contributions of the [Lab/Team Name] for access to the NVIDIA Jetson Orin Nano Developer Kit and associated infrastructure. Special thanks are extended to the industrial partners who provided facility access and anonymised production data for experimental validation. The authors thank the open-source communities behind Ultralytics YOLOv8, InsightFace, NVIDIA DeepStream, FastAPI, and PyTorch, whose publicly available tools and documentation made this work possible.

REFERENCES

- [1] T. H. Vo and G. S. Lee, "A vision-based method for automatic detection of personal protective equipment usage on construction sites," *Automation in Construction*, vol. 69, pp. 97–107, 2016.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE CVPR*, 2016, pp. 779–788.
- [3] W. Liu, D. Anguelov, D. Erhan et al., "SSD: Single shot multibox detector," in *Proc. ECCV*, 2016, pp. 21–37.
- [4] Y. Li, H. Wei, Z. Han, J. Huang, and W. Wang, "Deep learning-based safety helmet detection in engineering management," *J. Construction Engineering and Management*, vol. 146, no. 8, 2020.
- [5] N. D. Nath, A. H. Behzadan, and S. G. Paal, "Deep learning for site safety: Real-time detection of personal protective equipment," *Automation in Construction*, vol. 112, p. 103085, 2020.
- [6] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive angular margin loss for deep face recognition," in *Proc. IEEE CVPR*, 2019, pp. 4690–4699.
- [7] J. Guo and X. Zhu, "InsightFace: An open-source 2D and 3D deep face analysis library," *IEEE Trans. Biometrics, Behavior, and Identity Science*, vol. 3, no. 3, pp. 380–391, 2021.
- [8] E. Principi, F. Vesperini, S. Squartini, and F. Piazza, "Acoustic novelty detection with adversarial autoencoders," in *Proc. IJCNN*, 2017, pp. 3324–3330.
- [9] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger, "MVTec AD — A comprehensive real-world dataset for unsupervised anomaly detection," in *Proc. IEEE CVPR*, 2019, pp. 9592–9600.
- [10] K. Roth, L. Pemula, J. Zepeda et al., "Towards total recall in industrial anomaly detection," in *Proc. IEEE CVPR*, 2022, pp. 14318–14328.
- [11] J. Yu, Y. Zheng, X. Wang et al., "FastFlow: Unsupervised anomaly detection and localization via 2D normalizing flows," *arXiv:2111.07677*, 2021.
- [12] NVIDIA Corporation, "NVIDIA DeepStream SDK 7.0 Developer Guide," NVIDIA Technical Documentation, 2024.
- [13] M. Kaur and V. Monga, "Edge AI for real-time video surveillance using DeepStream SDK on NVIDIA Jetson devices," in *Proc. IEEE Int. Conf. Smart Computing*, 2021, pp. 1–6.
- [14] R. Babu, K. Subramaniam, and S. Ramesh, "Multi-camera vehicle detection and tracking using NVIDIA DeepStream on Jetson AGX Xavier," *J. Real-Time Image Processing*, vol. 19, no. 4, pp. 789–803, 2022.
- [15] S. Chen, Y. Liu, X. Gao, and Z. Han, "MobileFaceNets: Efficient CNNs for accurate real-time face verification on mobile devices," in *Proc. CCF Chinese Conf. Computer Vision*, 2018, pp. 428–438.