Edge-assisted human action recognition for video surveillance

Aibek Aitkozha
Department of Computer Science
Nazarbayev University
Astana, Kazakhstan
aibek.aitkozha@nu.edu.kz

Artur Iskakov
Department of Computer Science
Nazarbayev University
Astana, Kazakhstan
artur.iskakov@nu.edu.kz

Yerkebulan Kenzhebek
Department of Computer Science
Nazarbayev University
Astana, Kazakhstan
yerkebulan.kenzhebek@nu.edu.kz

Nguyen Anh Tu
Department of Computer Science
Nazarbayev University
Astana, Kazakhstan
tu.nguyen@nu.edu.kz

Abstract

The project addresses the significant challenges posed by the vast amount of video data generated by Internet of Things (IoT) devices, especially surveillance cameras, by developing an edge-assisted human action recognition system (HAR). Utilizing edge computing and deep learning technologies, including advanced pose estimation and convolutional neural networks (CNNs), the system aims to provide real-time HAR with minimal latency and reduced reliance on cloud resources. Key components include end devices for data capture, a cloud server for model training and management, and a web application for user interaction. This integration sets a new standard in real-time, edge-assisted video analytics by tackling traditional challenges related to latency, scalability, and efficiency. The project not only progresses through stages such as dataset creation, pipeline development, and software architecture design but also demonstrates the practical application and effectiveness of these technologies in enhancing video surveillance systems.

I. Introduction

The core challenge lies in proposing and implementing an effective solution that can handle the massive influx of video data from IoT devices, particularly surveillance cameras, with minimal latency. Traditional cloud-based processing frameworks are often overwhelmed by the volume and velocity of this data, resulting in unacceptable delays that compromise the functionality of real-time human action recognition (HAR) systems. This necessitates a novel approach that can efficiently process high volumes of data on-site without relying heavily on cloud computing resources.

Motivation:
The motivation for this project stems from the need to enhance the responsiveness and efficiency of video surveillance systems. By addressing the latency and scalability challenges, the system aims to improve monitoring applications that are critical for public safety, security, and operational efficiency.

Solution Overview:
Our solution leverages edge computing and advanced deep learning technologies, including convolutional neural networks (CNNs) and pose estimation, to develop an edge-assisted HAR system. This system is designed to perform real-time action recognition with minimal latency and reduced dependency on cloud resources. Key components of our system include end devices for robust data capture, a cloud server for centralized model training and management, and a web application to facilitate user interaction. Through strategic integration of these technologies, our project not only demonstrates the practical application but also sets a new standard in real-time, edge-assisted video analytics.

This report is organized into several sections, each aiming to provide a detailed view of the different aspects of our project:

1) Background and Related Work: The section describes the development process and challenges encountered in a project.
2) Project Approach: Details the technologies and frameworks used in our solution, explaining the rationale behind each choice and how they integrate to form our system.
3) Project Execution: Describes the process of implementing our edge-assisted HAR system, including the development of the DD-Net model, dataset preparation, and the setup of our edge computing environment.
4) Evaluation: Discusses the methodologies employed to assess the performance and effectiveness of our HAR system. This section includes a summary of the testing phases, the metrics used for evaluation, and the results

obtained.

5) Conclusion and Future Work: Summary of the project outcomes and the impact of our work. The discussion extends into potential future developments that could further enhance the system's performance, suggesting continuity and future engagement with the academic and professional communities.
6) References: Sources that informed our understanding and supported our claims throughout the report.

II. Background and Related Work

Human action recognition has become a pivotal technology in numerous applications ranging from security surveillance to patient monitoring systems. The integration of edge computing has further revolutionized this field by enabling real-time processing and responsiveness, essential in dynamic environments. This section reviews relevant technologies and methodologies that underpin our project, highlighting the evolution of action recognition systems and the role of edge computing.

Human Action Recognition

Traditionally, human action recognition has been implemented using Recurrent Neural Networks (RNNs) due to their ability to model temporal sequences effectively. While effective, RNNs often suffer from high latency and computational inefficiency when dealing with long video sequences (Yang, F. et al., 2019) [1]. Consequently, there has been a shift towards using Convolutional Neural Networks (CNNs) and pose estimation models which offer substantial improvements in speed and accuracy. Our project employs the DDNet model, a lightweight CNN that provides robust action recognition from skeletal data, offering a significant enhancement over traditional RNN-based approaches.

Edge Computing in Video Analytics

Edge computing facilitates data processing at or near the source of data acquisition, which in the context of video analytics, translates to immediate processing on edge devices such as cameras and IoT devices. This approach minimizes latency and bandwidth usage, which are critical for real-time applications. NVIDIA's Jetson Xavier AGX, an advanced edge computing device, serves as the backbone of our system, processing data directly from video inputs for immediate analysis.

Software and Tools

Our system leverages NVIDIA's DeepStream SDK, a comprehensive toolkit for building streaming analytics applications. DeepStream facilitates the creation of sophisticated processing pipelines that integrate neural networks for video, audio, and image understanding. For robust communication between the edge device and the cloud, we utilize Kafka for message queuing and Redis for data caching, ensuring efficient and reliable data flow.

During the development process, we encountered several challenges in implementing effective broadcasting solutions. Inspired by the approach of Gustavsson and Christensen [2], which advocated for direct server-client communication via WebRTC without relying on intermediate services, our system adopted WebRTC to enable real-time video streaming within our web application. Additionally, the work of Dian et al., which explored the use of the H.264 compression standard alongside various communication protocols, influenced our decision to implement RTSP for creating video sources [3]. This approach enhances the adaptability of our system to various user scenarios, providing flexible and efficient streaming solutions.

III. Project Approach

To implement our solution, we used a variety of ready-made technologies:

1) Docker: We used Docker to simplify deployment by packaging software components into containers, ensuring consistency and efficiency across different environments.
2) ZeroMQ: We chose ZeroMQ (zmq) for inter-communication between system components as it offers high-performance, lightweight messaging without the need for a dedicated message broker.
3) Kafka: We used Kafka for efficient and reliable handling of high-throughput, real-time data delivery between edge devices and servers, which is a common solution for storing and processing a continuous flow of data.
4) Redis: provides fast and low-latency data access that we use for edge-server communication in order to achieve the lowest possible latency.
5) Vue.js: was chosen as the frontend framework for its simplicity and our prior experience with the tool. Its component-based architecture facilitated modular development, while reactive data binding ensured dynamic and responsive user interfaces.
6) Centrifugo: was used for notifications due to its real-time capability, providing low-latency message broadcasting over WebSockets, which enhances user experience with immediate updates.
7) PostgreSQL: we chose Postgres due to its reliability, ACID compliance, and good support for advanced data types and SQL standards, making it highly reliable for complex applications.
8) MediaMTX: we decided to use MediaMTX for stream broadcasting over different protocols due to its ability to efficiently multiplex and manage multiple media streams.
9) YoloV8: The YoloV8 model was chosen for pose estimation due to its high-speed real-time processing capabilities and improved accuracy in detecting and localizing skeleton key points, even in complex visual environments.

10) Kubernetes: We chose Kubernetes for managing containerized applications due to its powerful orchestration capabilities, enabling automatic scaling, load balancing, and self-healing of applications across clusters.
11) GStreamer: we used this framework to create video streaming applications.
12) Nvidia Deepstream SDK: was used for video analytics pipeline as it optimizes processing through GPU acceleration, enabling real-time streaming analytics at scale with reduced infrastructure costs.

In order to minimize latency and distribute the load between the integral parts of the system, we have come up with the following architecture:

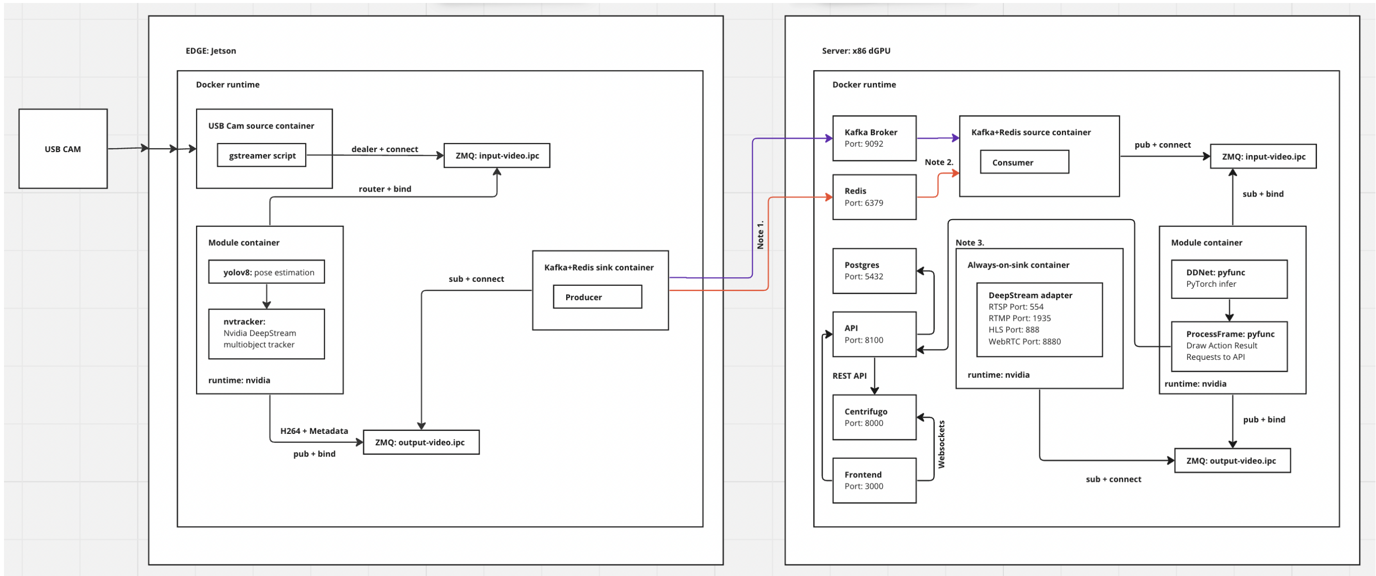


Fig. 1

There are 2 main components in our solution: Edge device and Server.
Both of them have a list of integral partitions
Edge device:

1) ZeroMQ instances:
For communication between integral components, we used ZeroMQ sockets. We've placed 2 zmq-sockets in the edge device:
Input socket - used for passing data from camera feed, RTSP source, and module component;
Output socket - used for providing data to Kafka&Redis sink;
2) USB cam source container
Docker container that we use to process the camera's USB feed with the usage of GStreamer. Within this container, the feed from the camera is encoded and sent to the input zmq-socket.
3) Module container
This component is built on top of DeepStreamSDK and used for estimating pose and extracting the human skeleton joint positions. This component accesses the camera feed from the input zmq-socket, decodes it, and sends extracted joint information to the output zmq-socket.
4) Camera Management API
API service written in Golang and designed for automatic deployment of an RTSP source container on an "add camera" call.
5) RTSP source container
Container that broadcasts frames to the input zmq-socket with usage of GStreamer framework.
6) Kafka-Redis sink container
The Kafka-Redis Sink Adapter sends video stream metadata to Kafka and frames content to Redis. Frame content location is encoded as `<redis-host>:<redis-port>:<redis-db>/<redis-key>`. `<redis-key>` is in the format `REDIS_KEY_PREFIX:UUID` where `UUID` is a unique

identifier of the video frame.

Server:

1) ZeroMQ instances:
For inter-communication in the server we also used zmq:
Input socket - used for passing the frame data from Kafka&Redis source container to Module Container;
Output socket - used for storing the resultant frames with the skeleton and action drawn on them;

2) Kafka broker and Redis instance
The components are responsible for receiving and storing each frame data sent by the edge device.

3) Kafka-Redis source container
The Kafka-Redis Source adapter container takes video stream metadata from Kafka and fetches frame content from Redis. Frame content location is encoded as ¡redis-host¿:¡redis-port¿:¡redis-db¿/¡redis-key¿.

4) Module Container
The container is also built on top of DeepStream SDK and responsible for fetching frame metadata from the input zmq-socket, analyzing the human skeleton join positions provided, and using the DD-net to classify on what action has been performed on given metadata. Then it draws the human skeleton on a frame and writes the label of action. Decodes the resultant frames and publishes them in output zmq-socket. It also sends requests to the backend service (API) when a certain action has been detected.

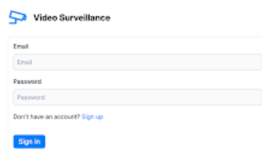
5) Always-on-sink container
The Always-On RTSP Sink Adapter streams the video through RTSP, LL-HLS, and WebRTC. This adapter consistently transcodes the incoming stream to maintain uninterrupted streaming, even if the source ceases operation. Under such circumstances, the adapter will continue to broadcast a static image until the source resumes data transmission. When the Nvidia Runtime is accessible, this adapter utilizes the DeepStream SDK to perform hardware-based transcoding and scaling of the incoming stream.

6) Postgres
Database for storing data about users, notifications, existing cameras, and their connections.

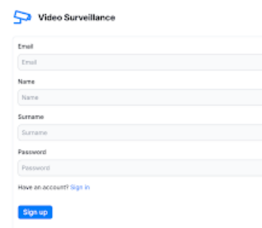
7) API
Backend service provides a list of endpoints for controlling users, cameras, and notifications.

8) Centrifugo
Centrifugo instance is being used for broadcasting notifications received from the Backend to the Frontend application via WebSockets.

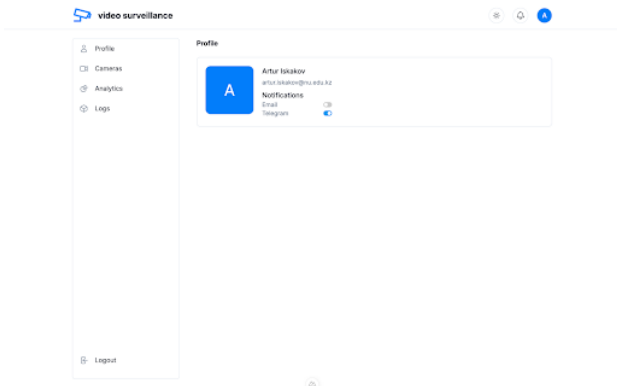
9) Frontend
Written in VueJs and provides a GUI for authentication, configuration of notifications, control over camera connections, and statistical graph observations:



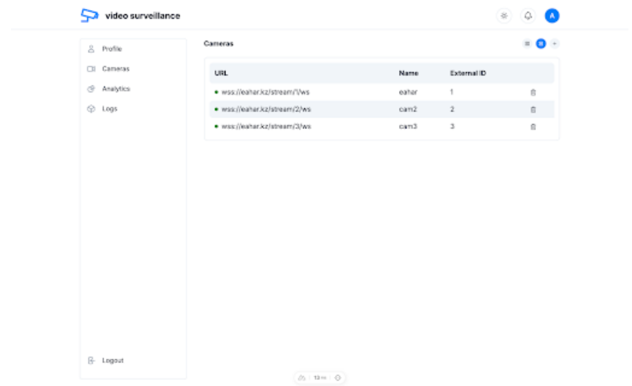
(a) Login page



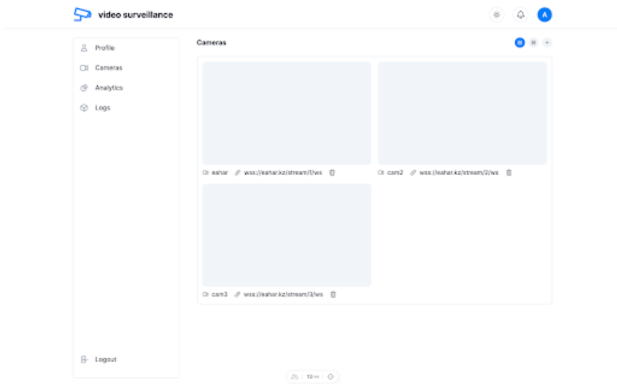
(b) Registration page



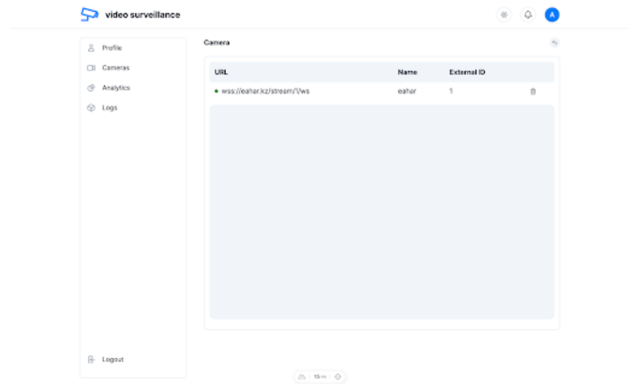
(a) Profile page



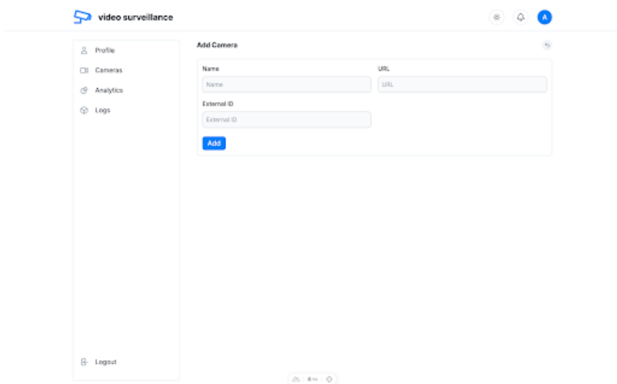
(b) Cameras list page



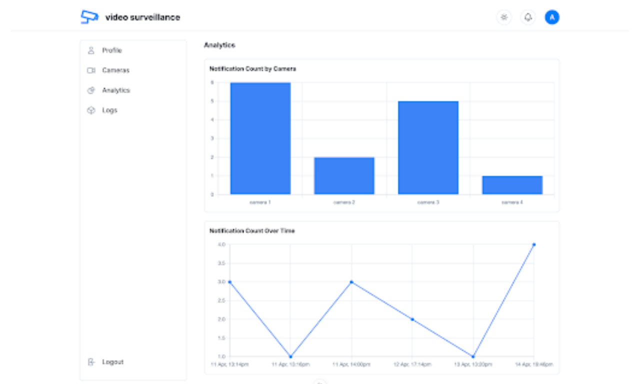
(a) Cameras grid page



(b) Camera details page



(a) Add camera page



(b) Analytics

IV. Project Execution

Over the last two semesters, our project focused on developing an edge-assisted HAR system to address the challenges posed by the high volume of video data from IoT devices.

Dataset Development
Initially, we realized the DD-Net on an edge device, starting with installing the necessary dependencies and integrating the network. To enhance our model, we collected a custom dataset. This dataset comprises 300 videos, including three distinct actions: hand-waving, boxing, and walking. Each action is captured in a variety of angles to enhance the model's ability to generalize. We collected 100 videos for each of chosen actions. We took 70 videos of each action type and added them to train sample, while other 30 videos of each type were used as a test stample.

Fig. 6

DD-net training and Pose Estimation
In order to start training DD-net we needed a pose estimation model that will preprocess the videos, extract joint positions from them and pass the extracted data to DD-net. During our testing phase, we evaluated multiple pose estimation models to identify the most effective solution for our system and ultimately decided to move on with the YoloV8 pose estimation.

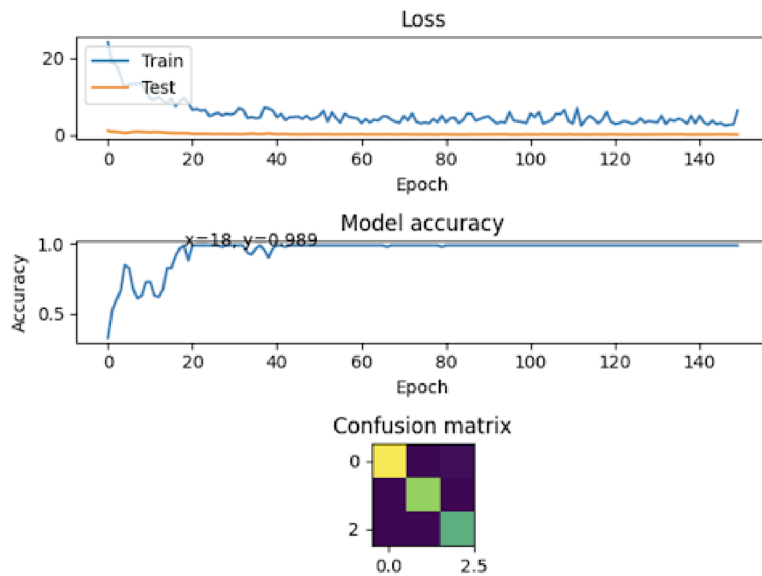


Fig. 7

We trained our DD-net with our custom dataset, ran some tests and were satisfied with the resultant performance. However, for the final evaluation and more sophisticated tests we retrained our DD-net model with the JHMDB dataset and achieved the accuracy of 77%.

Management

For our project, we've picked Notion as our project management tool, aligning with agile methodology. Notion's real-time collaboration and customizable workspace helped us organize tasks, track one-week sprints, and keep all project documentation in one place efficiently. Its ability to adapt to different project needs makes it a good fit for smooth project management.

Research and Architecture

Despite the successful training of DD-net and its incorporation with the YoloV8 pose estimation model, we encountered high latency during the initial trials on the NVIDIA Jetson device. This prompted us to revise our architecture by incorporating a server to distribute the workload between the edge device and the server, optimizing both the processing speed and system responsiveness. Extensive research was conducted to select the most effective technologies and practices for real-time video streaming and efficient data transfer between services.

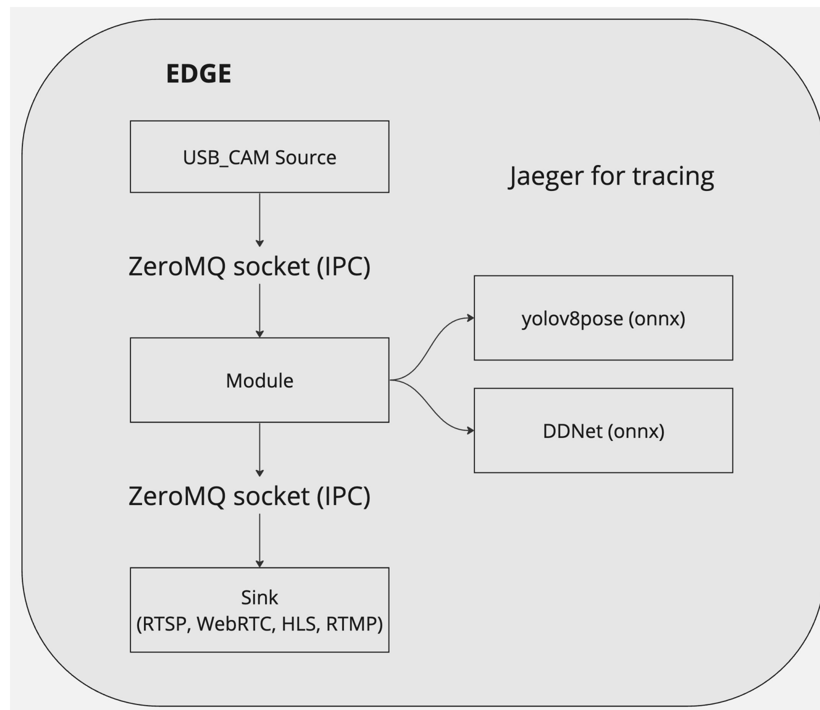


Fig. 8

During the development of our edge-assisted HAR system, we faced significant challenges in achieving effective load distribution for video processing. To address these challenges, it was imperative to explore and understand the prevailing methods used in video surveillance for data sharing and service communication. Consequently, we conducted extensive research on popular video surveillance architectures and their communication strategies. This investigation provided us with a wealth of knowledge that informed our architectural decisions, enabling us to experiment with different approaches and assess their efficiency.

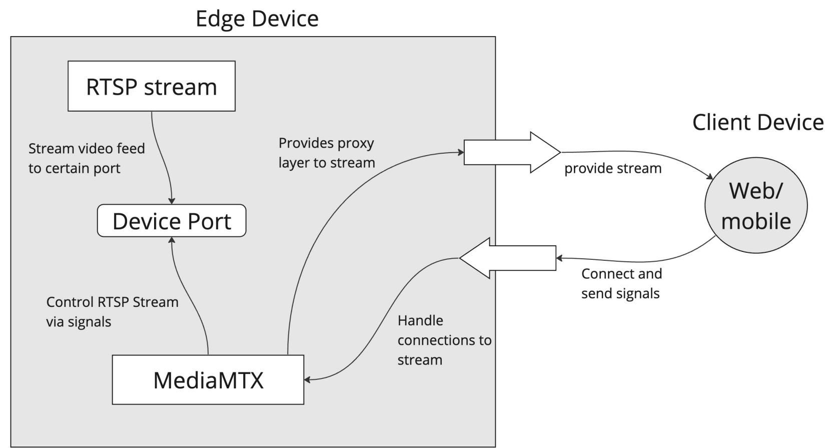


Fig. 9

After rigorous integration of various methodologies and testing of multiple technologies, we successfully devised a solution that met our expectations for low latency and high efficiency.

Backend and Frontend

We also designed and implemented the backend and frontend components to support the system's architecture effectively. The backend was structured through two distinct services written in GoLang, which communicated seamlessly with each other to fulfill the architectural needs. These services were crucial in handling CRUD operations and managing the flow of data across the system. On the frontend, we employed Vue.js to develop responsive web interfaces. This choice enabled us to provide dynamic visualization of the video streams and real-time notifications of recognized actions, enhancing user interaction and overall system usability. These integrated efforts ensured that both backend robustness and frontend user experience were optimized to meet the demands of our edge-assisted HAR system.

Evaluation

The final stages of the project were dedicated to system evaluation, where we experimented with various classifiers integrated with DD-Net to assess their impact on inference duration and overall latency. This evaluation helped us understand the performance bottlenecks and optimize our system accordingly.

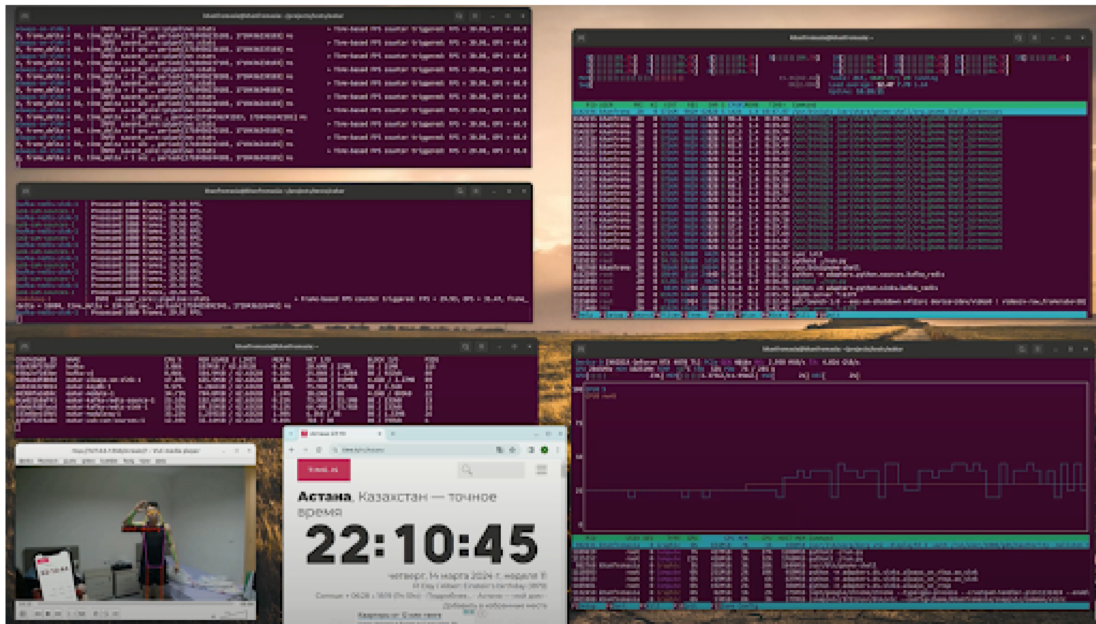


Fig. 10

In the evaluation phase of our project, considerable effort was dedicated to conducting extensive testing to measure system performance, focusing on latency, seconds per frame, and other key metrics. This process was both resource-intensive and time-consuming but essential for pinpointing system bottlenecks and areas that needed improvement. The insights obtained from these evaluations were critical in systematically refining our system. Through continuous testing and adjustments, we enhanced the effectiveness of our edge-assisted HAR system, demonstrating the importance of thorough and iterative evaluation in achieving optimal performance.

V. Evaluation

The evaluation of our edge-assisted human action recognition (HAR) system incorporated comprehensive testing using the JHMDB dataset, focusing on both model accuracy and inference performance across different computing environments. The JHMDB dataset was represented in 3 splits of train and test sets.
So the first split contained 660 train and 268 test samples:

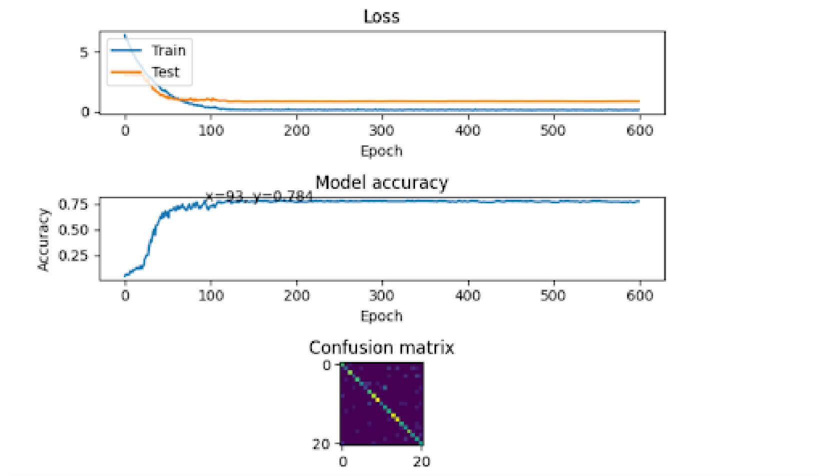


Fig. 11

With the highest accuracy(standalone DD-net) of 78.4%. The second split contained 658 train and 270 test samples:

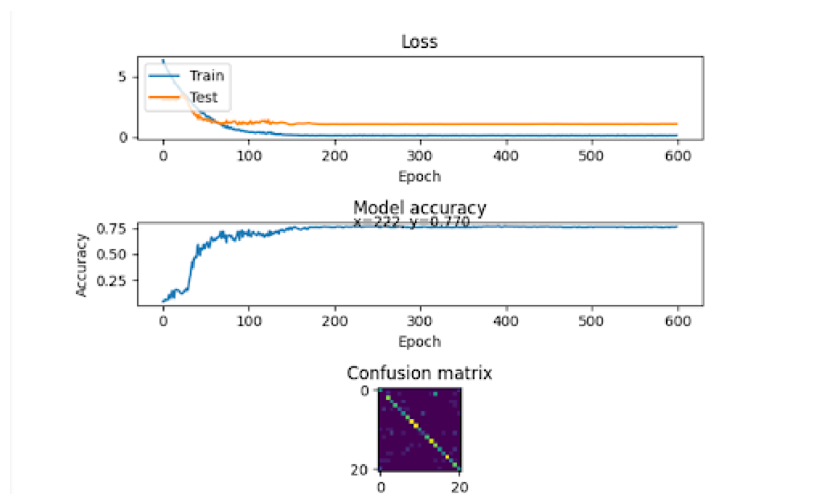


Fig. 12

With the highest accuracy(standalone DD-net) of 77%. The third split contained 663 train and 265 test samples:

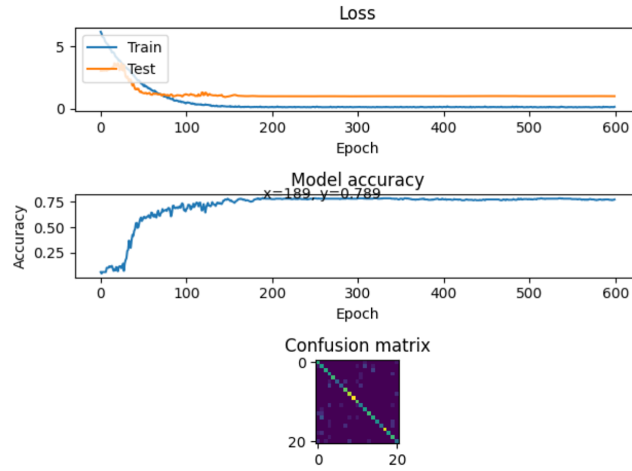


Fig. 13

With the highest accuracy(standalone DD-net) of 78.9%

Our accuracy testing involved three configurations: DD-net alone, DD-net embedding with a Support Vector Machine (SVM) classifier, and DD-net embedding with a Random Forest (RF) classifier. The results, as shown in Table 1, indicate that the DD-net embedding +RF achieved the highest accuracy at 77.22%. The incorporation of the SVM classifier and standalone DD-net resulted in a slight decrease in accuracy to 77.21% and 76.96% respectively. This suggests that the DD-net accuracy might be enhanced with the selection of proper classification solutions, however, the accuracy with other classifiers doesn't change dramatically. For the accuracy values, we calculated the means of average model testing accuracy.

TABLE I: Accuracy comparison on JHMDB

Model	#param	Accuracy
DD-net	DDNet: 1790565	77.21%
DD-net embedding + SVM	DDNet: 1790565 Number of Support Vectors: 203	76.96%
DD-net embedding + RF	DDNet: 1790565 Number of Trees: 100 Total Number of Nodes: 3192	77.22%

Inference performance was assessed on two types of devices: a high-powered local device and a more constrained edge device.
Local Device:
- GPU: RTX 4070TI
- CPU: 12th Gen Intel(R) Core(TM) i7-12700F
- RAM: 64 GB
- GRAM: 12GB
Edge Device:
- GPU: 512-Core Volta GPU with Tensor Cores
- CPU: 8-Core ARM v8.2 64-Bit CPU, 8 MB L2 + 4 MB L3
- RAM: 32 GB

TABLE II: Inference time on local device

Model	#inferences	#Frames	Total time	Inference per second	FPS
DD-net	89900	2876800	93.4907 sec	961.593	30770.98
DD-net embedding + SVM	89900	2876800	108.0295 sec	832.1801	26629.76
DD-net embedding + RF	89900	2876800	212.7462 sec	422.5692	13522.22

The local device tests (Table 2) showed that the standalone DD-net was the fastest, achieving an inference rate of 961.593 per second and a frames per second (FPS) rate of 30770.98 on the local device. The addition of SVM and RF classifiers slowed the inference speeds to 832.1801 and 422.5692 inferences per second, respectively, which directly impacted the FPS rates.

TABLE III: Inference time on edge device

Model	#inferences	#Frames	Total time	Inference per second	FPS
DD-net	89900	2876800	633.5781 sec	141.8925	4540.56
DD-net embedding + SVM	89900	2876800	768.0183 sec	117.0545	3745.74
DD-net embedding + RF	89900	2876800	2056.4742 sec	43.7156	1398.90

On the edge device (Table 3), all models demonstrated a significant drop in performance due to hardware constraints. The DD-net maintained the best performance relative to the other classifiers but showed a substantial decrease to 141.8925 inferences per second and 4540.56 FPS. The SVM and RF classifiers experienced more pronounced degradation, with the RF classifiers notably reducing to only 43.7156 inferences per second and 1398.90 FPS.

The data highlights the trade-offs between model complexity and computational efficiency, especially in edge-computing scenarios where resources are limited. While the DD-net alone provided the best balance of accuracy and speed on both device types. This is particularly evident in the edge device scenario, where the resource-intensive nature of SVM and RF classifiers resulted in noticeable slowdowns.

TABLE IV: Model latency on a local device

Model	Preprocessing (mean, msec)	Feature extraction (mean, msec)	Classification (mean, msec)	Total time (msec)
DD-net	0.05	0.84	0.06	0.95
DD-net embedding+SVM	0.05	0.86	0.21	1.12
DD-net embedding+RF	0.09	0.89	1.33	2.32

Table 4 details the latency measurements across the stages of preprocessing, feature extraction, and classification for each model configuration on a local device:
DD-net showed the lowest latency with a total time of 0.95 milliseconds (msec), demonstrating its efficiency in handling the entire process swiftly.
DD-net with SVM classifier experienced a slight increase in total latency to 1.12 msec. The notable rise was in the classification stage, where the time increased from 0.06 msec to 0.21 msec, reflecting the additional computational overhead introduced by SVM.
DD-net with RF classifier had the highest latency at 2.32 msec. The classification stage saw a significant increase to 1.33 msec, indicating that RF's computational complexity considerably affects performance.

TABLE V: Model latency on an edge device

Model	Preprocessing (mean, msec)	Feature extraction (mean, msec)	Classification (mean, msec)	Total time (msec)
DD-net	0.27	5.89	0.76	6.92
DD-net embedding+SVM	0.35	6.18	1.39	7.92
DD-net embedding+RF	0.90	6.34	13.44	20.68

Table 5 examines the same model configurations but evaluates their performance under the constrained resources of an edge device:
The latency results from both devices highlight the performance trade-offs associated with integrating more complex classifiers like SVM and RF with the DD-net model. While the local device could accommodate the increased demands of SVM and RF with relatively minor impacts on latency, the edge device showed significant performance degradation, particularly with the RF classifier.
Another important parameter to evaluate was a performance measurement of the DD-net standalone model in handling different scenarios based on the number of individuals present in the video streams. To achieve this, we selected a series of video samples, each varying in resolution, size, and frame rate, but specifically categorized by the number of persons appearing simultaneously—ranging from one to eight. These videos were used to systematically assess the impact of both the number of streams and the number of persons on the model's inference latency and the seconds required per frame on a local device. This comparative approach allowed us to understand the scalability of the DD-net model under increasing complexity and to identify potential limits in its capability for real-time human action recognition, particularly in densely populated video scenes.

1 person:
- type: video/mp4 720x1280
- size: 3.8 MB
- duration: 13.54 s
- Frame rate: 25

2 person:
- type: video/mp4 640x360
- size: 3.8 MB
- duration: 10.97 s
- Frame rate: 23.976

5 person:
- type: video/mp4 640x360
- size: 1.073 MB
- duration: 5.56 s
- Frame rate: 25

8 person:
- type: video/mp4 640x360
- size: 1.5 MB
- duration: 9.96 s
- Frame rate: 25

TABLE VI: Impact of #streams and #persons on the inference latency in local device

	Latency for 1s	Latency for 2s	Latency for 3s	Latency for 4s	Latency for 5s
1p	0.0166 sec	0.0235 sec	0.0350 sec	0.0458 sec	0.0559 sec
2p	0.02645 sec	0.0432 sec	0.0631 sec	0.0802 sec	0.0983 sec
5p	0.05214 sec	0.0996 sec	0.1246 sec	0.1954 sec	1.1673 sec
8p	0.0738 sec	0.1324 sec	0.5554 sec	3.2366 sec	6.6184 sec

The performance experiments conducted on the DD-net standalone model with varying numbers of persons in the video streams on a local device show significant insights into the model's scalability and real-time processing capabilities. As detailed in Table 6, the latency increases markedly with the number of persons in the video. For videos with one and two persons, the increase in latency over five seconds is relatively linear and manageable, suggesting that the model handles fewer subjects with better efficiency. However, for videos containing five and especially eight persons, the latency escalates dramatically, particularly after the third second, indicating a substantial decrease in performance as the model processes more complex scenes with multiple individuals.

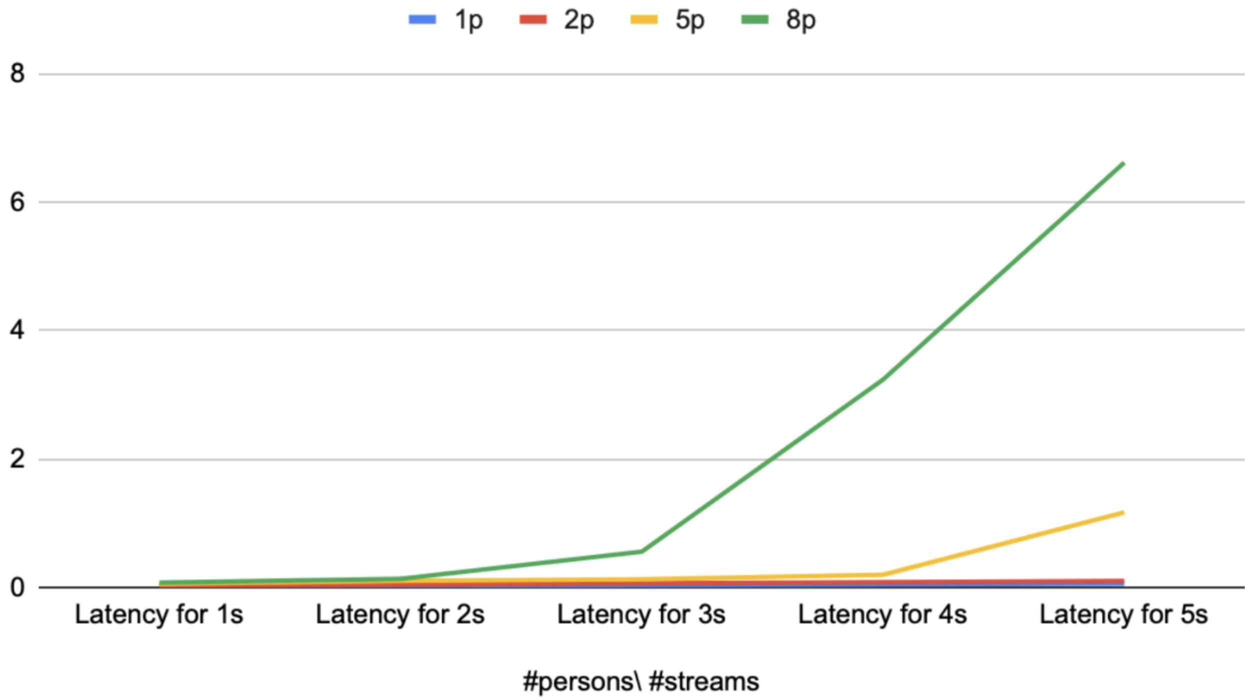


Fig. 14

TABLE VII: Impact of #streams and #persons on the inference seconds per frame in local device

	sec/frame for 1s	sec/frame for 2s	sec/frame for 3s	sec/frame for 4s	sec/frame for 5s
1p	0.0401 sec	0.0401 sec	0.0401 sec	0.0401 sec	0.0402 sec
2p	0.0264 sec	0.0418 sec	0.0419 sec	0.0420 sec	0.0420 sec
5p	0.0404 sec	0.0407 sec	0.0409 sec	0.0414 sec	0.0484 sec
8p	0.0403 sec	0.0405 sec	0.0422 sec	0.0530 sec	0.0667 sec

The seconds per frame data from Table 7 supports these findings. The frame processing time remains consistent for videos with one person regardless of the stream's length, maintaining around 0.0401 seconds per frame. However, as the number of persons increases, while the initial seconds per frame remains stable, there is a noticeable degradation in processing speed in more populated videos (5 and 8 persons), especially as the duration extends beyond three seconds. For the eight-person video, the seconds per frame more than double by the fifth second. This degradation reflects the increased computational burden imposed by multiple subjects, leading to longer processing times per frame, which can impact real-time application performance.

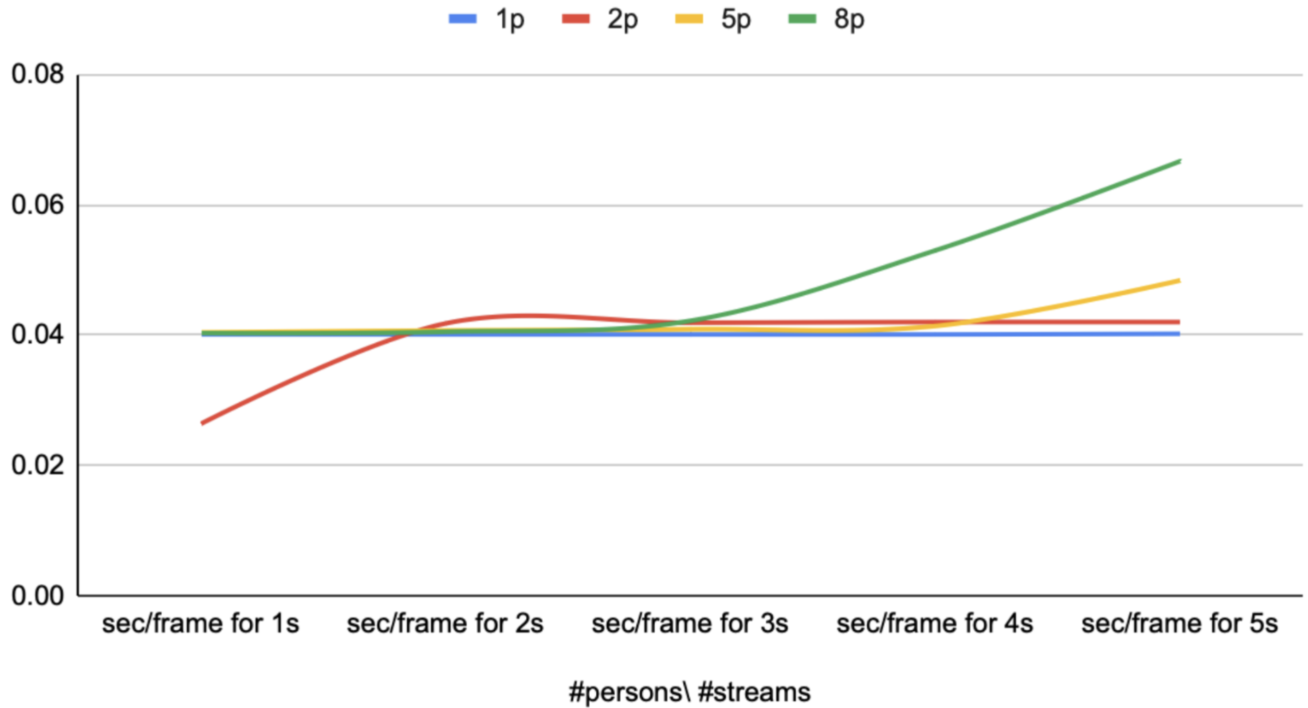


Fig. 15

The same experiment was conducted on an edge device.

TABLE VIII: Impact of #streams and #persons on the inference latency in the edge device

	Latency for 1s	Latency for 2s	Latency for 3s	Latency for 4s	Latency for 5s
1p	0.2994 sec	10.6092 sec	21.4768 sec	30.9822 sec	41.3507 sec
2p	5.8382 sec	18.8700 sec	31.7397 sec	45.1986 sec	58.5522 sec
5p	8.1650 sec	17.6127 sec	27.5082 sec	37.8781 sec	47.8628 sec
8p	26.5756 sec	57.7842 sec	88.9997 sec	118.6557 sec	154.4855 sec

The experiment conducted on the edge device reveals a significantly heightened inference latency and per-frame processing time compared to the local device, particularly as the number of persons in the video streams increases. Table 8 demonstrates a drastic increase in latency for all configurations on the edge device, with especially severe spikes for higher person counts, such as in the 8-person scenario where latency reached over 154 seconds at the five-second mark. This starkly contrasts with the relatively more stable latency growth observed in the local device settings, where the model managed to maintain lower latency even with increasing complexity.

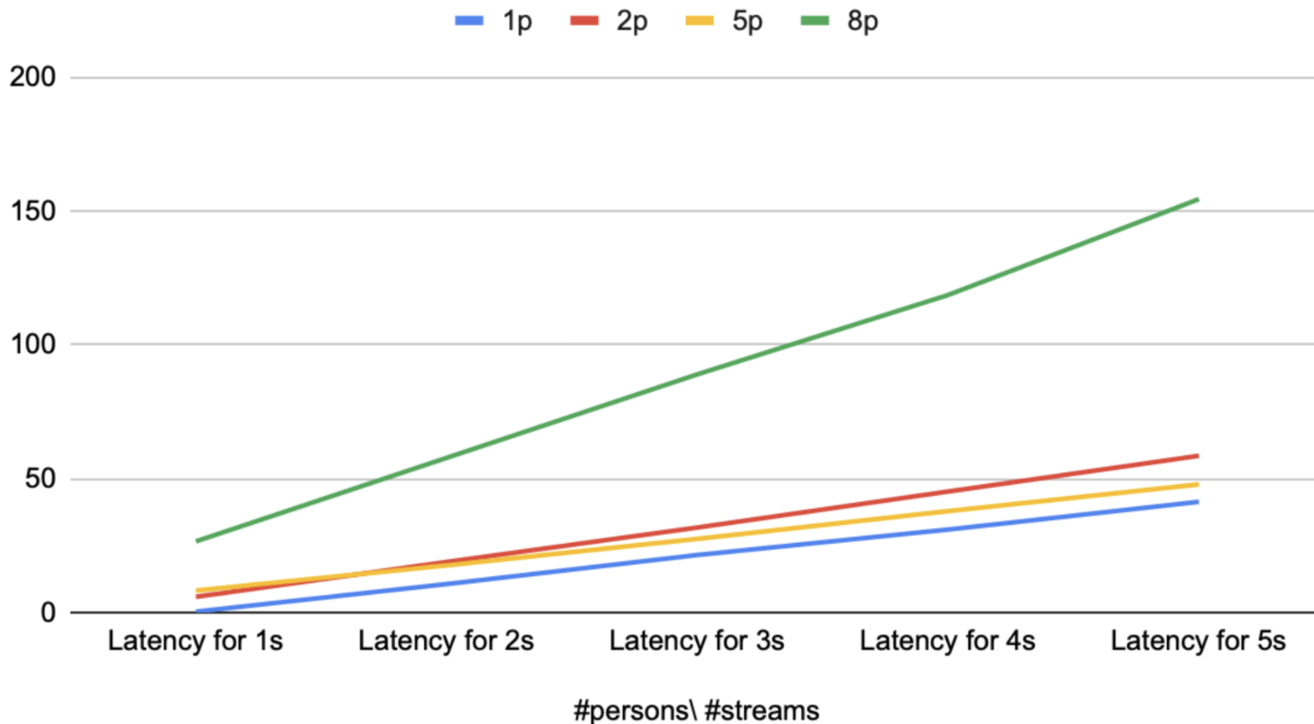


Fig. 16

TABLE IX: Impact of #streams and #persons on the inference seconds per frame in the edge device

	sec/frame for 1s	sec/frame for 2s	sec/frame for 3s	sec/frame for 4s	sec/frame for 5s
1p	0.0409 sec	0.07149 sec	0.1037 sec	0.1319 sec	0.1627 sec
2p	0.0639 sec	0.1136 sec	0.1627 sec	0.2140 sec	0.2649 sec
5p	0.0992 sec	0.1677 sec	0.2394 sec	0.3145 sec	0.3870 sec
8p	0.1472 sec	0.2730 sec	0.3990 sec	0.5185 sec	0.6631 sec

Similarly, the seconds per frame data from Table 9 shows a substantial increase across all configurations on the edge device, suggesting that each frame takes longer to process as the number of persons increases. For instance, in the one-person scenario, the time per frame increases from 0.0409 seconds to 0.1627 seconds over five seconds, while the eight-person scenario sees an initial frame time of 0.1472 seconds escalating to 0.6631 seconds. This degradation in performance is far more pronounced than on the local device, indicating that the edge device struggles significantly under the computational demands imposed by more complex video streams, affecting its suitability for real-time action recognition applications in dense scenarios.

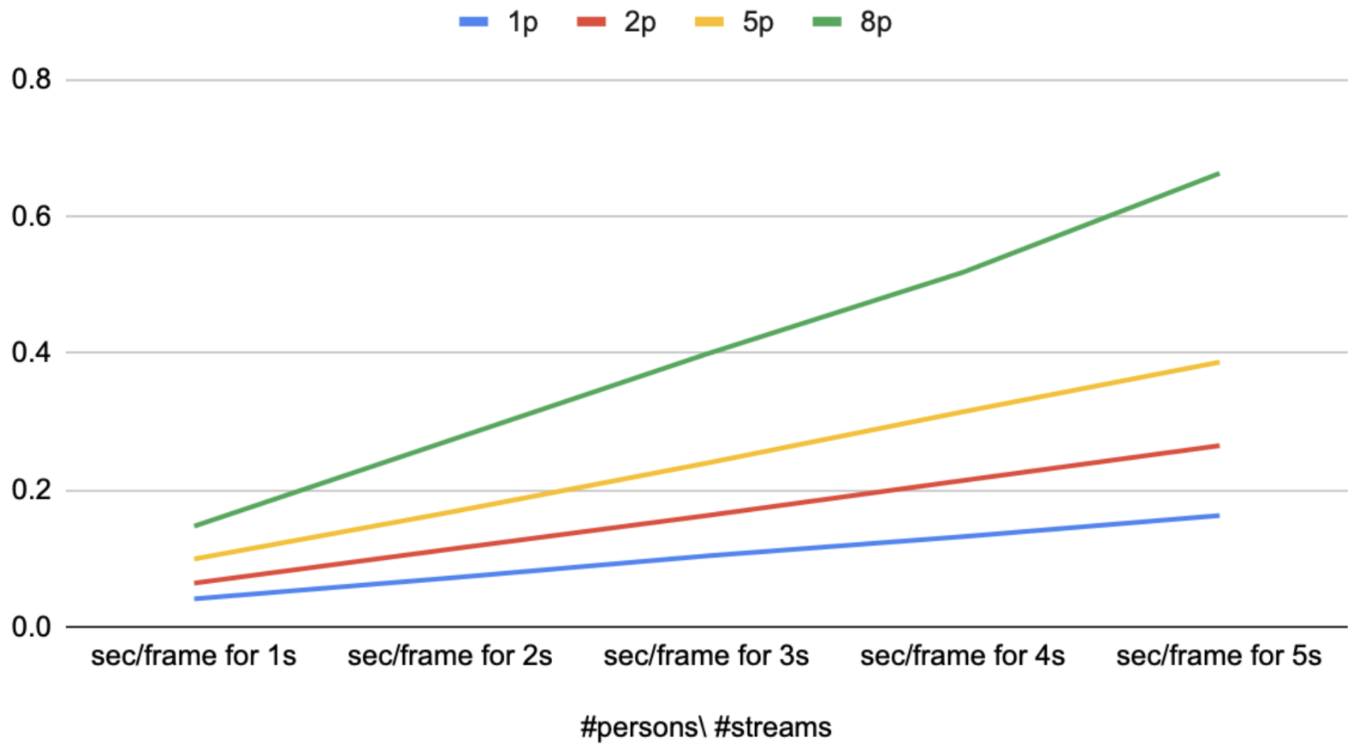


Fig. 17

VI. Conclusion and possible future work

Our comprehensive evaluation of the edge-assisted human action recognition (HAR) system using the DD-net model has provided valuable insights into its operational dynamics and performance across varying conditions. The results from extensive testing clearly illustrate the model's strengths and areas needing improvement, particularly in handling multiple streams and different numbers of individuals in real-time video surveillance.

Model Latency and Performance:

The standalone DD-net model demonstrated the best performance in terms of both accuracy and inference speed on less complex scenarios, making it highly suitable for edge devices. However, as the complexity of the scenarios increased—with more individuals in each frame—the latency and processing times escalated significantly. This is especially noticeable in edge computing environments where computational resources are more constrained than in local devices. The marked increase in latency and per-frame processing times with higher person counts underscores the need for models that are not only accurate but also efficient in resource utilization for edge deployment.

Impact of Scene Complexity:

The experiments highlighted a clear dependency of latency on the number of streams and the number of persons captured, with performance degrading as these variables increased. This degradation was more pronounced on edge devices than on local devices, suggesting that the current model architecture may need to be simplified or further optimized for scenarios expected to handle dense crowd scenes to avoid performance bottlenecks.

Strategic Implications for Future Development:

These findings are critical for guiding the next steps in the development of our HAR system. They suggest that while the DD-net model serves well under conditions with fewer individuals, alternative strategies may need to be considered for more complex applications. This could involve exploring more sophisticated machine learning techniques, software architectures, and hardware based support acceleration that maintain accuracy without compromising the speed required for real-time processing or considering a hybrid approach that dynamically adjusts the computational strategy based on the current load and scene complexity.
The experiments highlighted a clear dependency of latency on the number of streams and the number of persons captured, with performance degrading as these variables increased. This degradation was more pronounced on edge devices

than on local devices (server), suggesting that the current model architecture may need to be simplified or further optimized for scenarios expected to handle dense crowd scenes to avoid performance bottlenecks. Moving forward, continuous optimization of the model architecture will be essential to enhance its scalability and efficiency. By focusing on developing reliable HAR systems that meet the real-time processing requirements of modern edge deployments, we aim to provide effective surveillance solutions that are both responsive and resource-efficient.

References

[1] F. Yang, S. Sakti, Y. Wu, and S. Nakamura, "Make skeleton-based action recognition model smaller, faster and better," 2019.

[2] J. Gustavsson and H. Christensen, "Webrtc for peer-to-peer streaming from an ip camera," 2019. Student Paper.

[3] D. Chu, C.-h. Jiang, Z.-b. Hao, and W. Jiang, "The design and implementation of video surveillance system based on h.264, sip, rtp/rtcp and rtsp," in *2013 Sixth International Symposium on Computational Intelligence and Design*, IEEE, Oct. 2013.