# FLEX: An Embedded Pose Detector for Real-Time Fitness Coaching on Zynq UltraScale+

Oliver Quin
oliver.quin@mail.polimi.it
*Politecnico di Milano - Dipartimento di Elettronica, Informazione e Bioingegneria*

***Abstract*—Regular physical activity is associated with improved health outcomes, yet access to structured fitness environments remains limited: in 2023, fitness-club penetration in Europe reached only 10.1% of the population aged 15 and over. Automated coaching systems can partially address this gap by providing exercise feedback outside professional training environments. However, existing vision-based approaches rely on cloud computing or power-hungry computing platforms, simultaneously increasing latency, energy cost, and exposure of sensitive data. To address this gap we present FLEX, an edge-cloud solution for automated exercise coaching, with a focus on the squat. FLEX performs person detection, pose estimation, and posture analysis locally, leveraging the AMD Kria KV260 and its Deep Learning Processor Unit implemented on the FPGA fabric to accelerate neural network computations.**

**Across ten independent 30-minute runs, the edge pipeline processed 1,050,295 frames and sustained 58.35 $\pm$ 0.14 FPS, with a median end-to-end latency of 17.30 ms, a p99 latency of 20.71 ms, and an average energy cost of 0.103 $\pm$ 0.0003 J/frame.**

***Index Terms*—edge computing, embedded AI, pose estimation, exercise coaching, AMD Kria KV260, deep learning, DPU, edge-cloud architecture, large language model, real-time inference**

## I. Introduction

Physical activity plays a central role in long-term health, yet access to professional fitness environments remains limited. In 2023, fitness-club penetration in Europe reached only 10.1% of the population aged 15 and over [1], indicating that structured training environments reach only a minority of the population. The development of automated coaching systems capable of providing users with exercise guidance could increase fitness levels in the general population while ensuring sufficient guidance to minimize injury risk. Existing coaching solutions rely on two kinds of approaches to analyze movement: wearable sensors and Computer Vision (CV) technology. Wearable systems provide useful information, but they require specialized devices to be worn by the user, introducing friction in terms of comfort and adoption, especially for non-expert users. Conversely, vision-based systems offer a promising alternative, since they can estimate body posture without requiring any device attached to the body. This makes camera-based pose estimation a practical foundation for automated coaching applications. While convenient for user comfort, modern CV solutions based on Machine Learning (ML) techniques are particularly power-hungry and computationally demanding; this makes deployment on embedded hardware particularly challenging, especially when, alongside the CV ML models, Large Language Models (LLMs) are used for natural language interaction with the trainee. A cloud-based solution would provide additional resources but would require transmitting the video stream to remote providers, increasing latency and exposing sensitive visual information to third parties.

In this work, we present FLEX, an edge-cloud system designed for real-time exercise coaching on the AMD Kria KV260. Thanks to the heterogeneous nature of the target platform, we achieved 0.103 $\pm$ 0.0003 J/frame at 58.35 $\pm$ 0.14 FPS by running the CV models on AMD's DPU implemented on the FPGA. To meet real-time performance requirements, we leverage cloud computing power to offload the demanding LLM inference through a solution that allows us to maintain user privacy, effectively creating a privacy-aware edge-cloud partitioning strategy.

## II. Related Work

### A. Automated Exercise Coaching

Advancements in sensors and camera technologies have driven growing interest in automated coaching systems. In [2], early systems were classified into three categories: smartphone applications, wearable sensor devices, and image-processing-based systems. Early designs utilized the Microsoft Kinect due to computational limitations; in fact, by leveraging infrared depth sensing, researchers were able to simplify skeletal tracking and enable real-time movement analysis on hardware with limited capabilities. [3] further explored the capabilities of Kinect-based systems; the work compared live data against a database of references to assess exercise correctness in real time.

Once ML-based CV frameworks such as MediaPipe became popular, more software-based approaches arose, making general-purpose hardware an interesting option. In [4], researchers trained an ML classifier for exercise recognition and repetition counting, training the network on camera inputs with extracted keypoints. On shoulder exercises, their approach reached up to 99.80% classification accuracy.

Interest in friendlier human interfaces has also caught the attention of researchers. In [5], a cheap smart mirror implementation of MediaPipe-based pose estimation with an ML classifier performed the computation on a Raspberry Pi. However, no performance metrics were reported, making the

quality of the results difficult to quantify; nonetheless, this idea demonstrates a clear interest in different target hardware.

These works demonstrated feasibility and interest in automated exercise coaching systems. The principal focus was on exercise recognition and movement assessment; less attention has been placed on system-level challenges of deploying such functionality on resource-constrained edge platforms while maintaining real-time performance.

### B. Neural Network Deployment on Embedded Hardware

Deploying neural networks on embedded hardware presents an additional set of challenges that depend on the target platform. In general, limited computing power and power budget require careful design trade-offs between hardware and software.

To expand domains in which embedded devices can be used, heterogeneous architectures have been developed covering a wide scope of use cases. These accelerator-based architectures usually take the form of a SoC with a dedicated unit to perform task-specific computations. In [6] [7], different targets have been compared, concluding that the FPGA-based SoC of the AMD Kria KV260 presented a 163-fold increase in inference rate compared to a classical embedded target, the Raspberry Pi 4, and a 6-fold increase with respect to a GPU-based accelerator, the NVIDIA Jetson Nano.

The Kria KV260 leverages its FPGA to increase computational power. AMD provides a set of IP cores tailored for its ecosystem; among those, the Deep Learning Processor Unit (DPU) is of particular interest for workloads based on convolutions. The available configurations utilize 8-bit quantization. The impacts have been explored by [8], which showed that a ResNet-18 trained on brain tumor detection can achieve up to 192 FPS with approximately 5 W of board-level power consumption, proving that real-time vision approaches are deployable on the selected target.

Designing a multi-stage pipeline with frequent communication between the Cortex processor and the DPU with multiple models running is not trivial. [9] did exactly so, combining a two-stage person detection with depth-based pose estimation; they developed a fall detection system with 1.17 ms of total inference time and an accuracy of 72.86%. This work also presented another relevant result for our constraints: it is possible to run a complete pose detection pipeline in real time without relying on external computing infrastructures.

These works demonstrate the viability of the AMD Kria KV260 for real-time embedded inference under tight power and resource constraints, while keeping sensitive data confined to the device. However, local hardware alone cannot satisfy all computational demands: current LLM inference remains prohibitively expensive for the platform under the target real-time and power constraints. [10] and [11] demonstrate that running Qwen2.5-0.5B on the KV260 saturates both CPU and FPGA resources, achieving at best 5.1 tokens/s with a dedicated accelerator. This makes co-execution alongside a real-time pose estimation pipeline impractical under the intended deployment conditions. This constraint motivates a hybrid edge-cloud architecture, where sensitive inference runs locally while high-level feedback generation is offloaded to a remote server.

### C. Edge-Cloud Architectures for AI Inference

The growing complexity of deep learning (DL) workloads has driven significant research interest in hybrid architectures. In those designs, computation is partitioned between edge devices and remote servers. [12] provided a comprehensive taxonomy of distributed inference models; architectures were classified from pure on-device solutions to edge-cloud co-inference, where each computational level is assigned tasks that match available resources.

The enabler of edge inference is the adoption of domain-specific hardware accelerators; in this scenario, FPGA-based systems with high reconfigurability and high energy efficiency are particularly suited for DNN workloads [12]. But the problem of efficiently performing AI inference cannot be resolved just by hardware accelerators. [13] presented Edgent, which is a framework developed with the goal of efficiently partitioning computations across different layers of a mobile-edge server architecture, basing the offloading decision on available bandwidth and capabilities.

However, not every AI workload can be efficiently executed at the Edge. Large Language Models require many hardware resources to be properly run, often exceeding the capabilities of embedded platforms. Addressing this issue, [14] proposed an active strategy for cloud-edge inference offloading using a request system between the system actors to reduce latency while satisfying resource constraints.

Previous work shows that vision inference for real-time applications can be effectively deployed on edge devices such as the Kria KV260. Based on this evidence, we adopt a partitioned architecture for FLEX, keeping the vision pipeline on the edge while delegating feedback generation to the cloud.

## III. System Architecture

We developed a two-tier architecture following an edge-cloud approach. As the edge device, an AMD Kria KV260 performs real-time pose estimation via a quantized neural network deployed on the DPU; confining video processing to the edge allows raw video to remain on the local device. On the cloud side, a remote server acts as the Large Language Model inference provider and is responsible for feedback generation. The decision to offload LLM inference to a remote server rather than running it on the board is motivated by two primary factors: token generation speed and resource requirements [10], [11].

### A. Edge Pipeline

The edge pipeline is a multithreaded application designed to leverage the full computing power of the target platform. We implemented an event loop capped at 60 Hz to handle asynchronous work; worker threads are responsible for frame capture, LLM communication, and frame rendering.

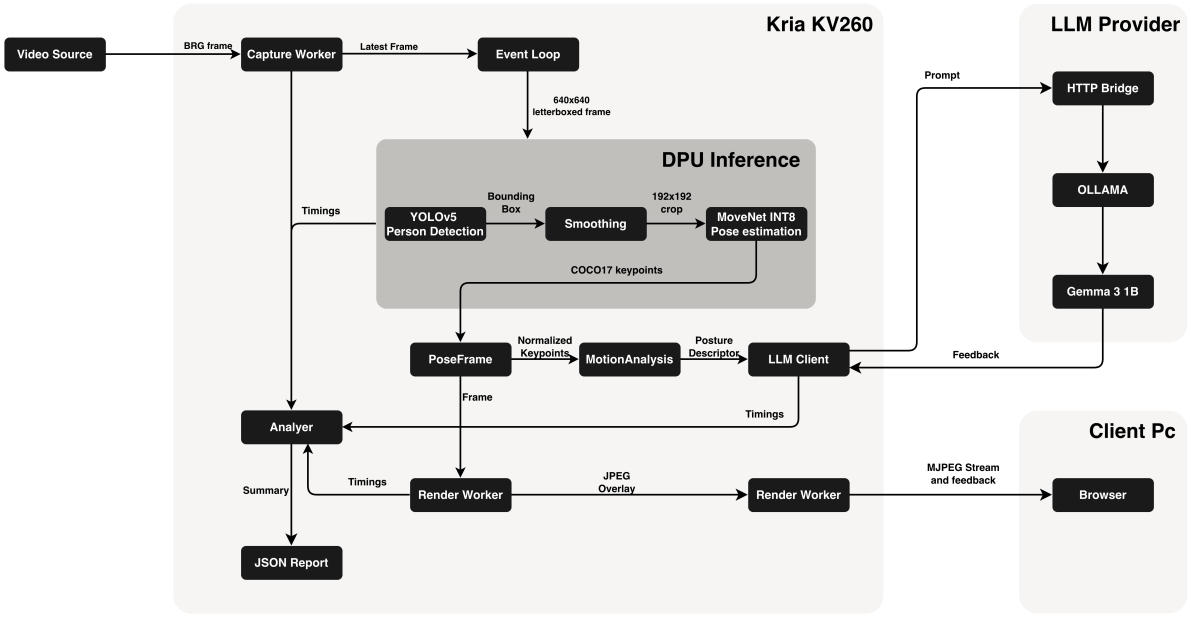



Fig. 1. Overview of the FLEX edge-cloud architecture. The central block represents the AMD Kria KV260, where the real-time perception pipeline is executed locally. Frames are acquired by the capture worker and processed by the main event loop, which invokes the DPU inference pipeline composed of YOLOv5-nano person detection, bounding-box smoothing, and MoveNet INT8 pose estimation. The resulting `PoseFrame` is used for local motion analysis, producing compact posture descriptors. These descriptors are sent to the LLM provider on the right, where an HTTP bridge forwards the request to Ollama and Gemma 3 1B generates textual coaching feedback. The client PC receives the MJPEG stream and the latest feedback through the browser interface, while the analyzer records timing information and exports a JSON summary for offline evaluation.

*1) Capture Worker:* The CaptureWorker thread implements the video interface, acquiring BGR frames either from a V4L2 camera device or from a file source via GStreamer. To decouple frame acquisition from the main event loop, a single-slot buffer has been implemented; the worker continuously overwrites the buffer with the last captured frame, while the main loop retrieves it, all via a non-blocking API. This design ensures that the main loop performance is independent of camera specifications and avoids stalls in the inference pipeline.

*2) DPU Inference:* The inference is implemented as a three-stage pipeline. The first stage runs YOLOv5-nano, which performs person detection; afterwards, MoveNet INT8 performs pose estimation, and lastly the resulting inference data is processed. In particular, YOLO produces a bounding box, which is smoothed via Exponential Moving Average; the frequency at which this stage is performed is a software knob for system performance: every `detect_interval` frames a new call to the model is made. On the frame crop bounded by the YOLO box, MoveNet INT8 is called; it processes a 192×192 region, producing seventeen keypoints in COCO format with coordinates normalized in the [0,1] range. The complete inference pipeline outputs a structured pose representation containing the normalized keypoints, the bounding box, a monotonic timestamp, and the relative profiling information.

*3) Motion Analysis:* After pose estimation, the system runs the motion analysis stage. From the computed `PoseFrame`, we extract a set of postural features. In particular, normalized coordinate differences are used to derive human-readable labels for squat depth, shoulder symmetry, and stance width. These labels constitute the only information that is permitted to leave the board perimeter. The remote LLM therefore receives only derived posture descriptors rather than video frames or raw keypoint coordinates. This approach reduces the amount of identifiable information exposed to the remote service while allowing natural-language coaching feedback to be generated remotely.

*4) Render:* The same asynchronous worker pattern used for the CaptureWorker is applied to the RenderWorker. The main event loop submits the latest BGR frame and its associated `PoseFrame` via a non-blocking interface, regardless of whether rendering or encoding are completed. The RenderWorker uses OpenCV to draw the pose overlay, including the bounding box, the projected pose skeleton, and a minimal HUD showing system performance. Skeleton rendering can be disabled via the `DRAW_POSE` compile-time flag. The MJPEG server exposes the processed video stream over HTTP alongside the rest of the user interface, implemented as a web application. This channel operates independently from the LLM service and does not contribute to real-time pipeline timing due to its asynchronous nature.

### B. Cloud Offloading

The remote server has access only to the condensed posture description, and its sole responsibility is the generation of coaching feedback based on the received information.

*1) Remote Ollama Server:* The remote server runs a small Python application acting as a bridge between the HTTP

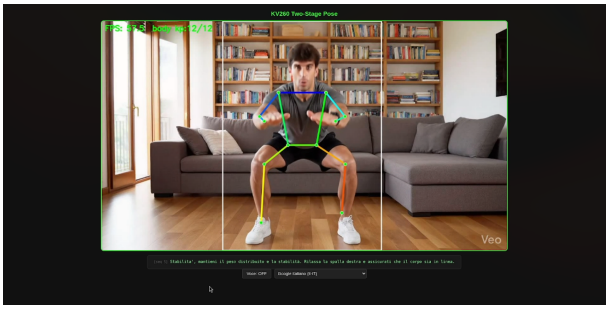



Fig. 2. FLEX User Interface.

interface and the inference backend. We use Ollama as the inference provider, which exposes an API to perform calls to the models. The selected model is Gemma 3 1B, a compact open-weight model from Google's Gemma 3 family. Its small parameter count makes it suitable for deployment on consumer hardware; nonetheless, as documented in the model card [15], [16], its multilingual capabilities and strong instruction following make it a good choice for generating personalized training feedback.

*2) Connectivity:* The connectivity between the board and the remote host is established via a reverse SSH tunnel. The Python bridge listens on `localhost:8080` of the remote machine; the tunnel maps `127.0.0.1:8080` on the KV260 via remote port forwarding. This allows the edge application to access the remote model through a local HTTP endpoint without additional network configuration.

*3) LLM Client:* On the edge side, the LLMClient runs in a dedicated thread and waits for submissions from the main loop, synchronized via a condition variable; this keeps network communication outside the real-time path. Requests are throttled with a 7 s minimum interval between consecutive submissions, preventing redundant prompts from saturating the remote server during operation. This value was chosen empirically to match the average prompt-to-feedback time, including the web interface's text-to-speech playback. At each iteration, the event loop retrieves the last unconsumed feedback via a non-blocking call and forwards it to the MJPEG server for browser display. In case of error, the previous message is retained; if none is available, a fallback error message is presented.

### C. Monitoring and Observability

We included an Analyzer component used to collect and aggregate measurements reported in the experiments. Inference, render submission, and end-to-end latency are recorded on a per-frame basis; every 60 frames the Analyzer aggregates throughput, mean latency per stage, loop overruns, dropped frames, LLM responses, and error counts. Session summaries are written in JSON format at the end of each run.

## IV. Implementation

### A. Target Platform and Software Stack

The target platform is an AMD Kria KV260, an embedded development board of the Zynq UltraScale+ MPSoC family. FLEX leverages the heterogeneous architecture by running control logic on the quad-core ARM Cortex, while offloading the inference pipeline to the DL Unit on the Programmable Logic.

The system runs the Vitis AI PetaLinux image provided by AMD. The control logic interacts with the DPU via the Vitis AI stack: `.xmodel` files are loaded via the VART runner, while XRT manages communication with the DPU overlay. This allows YOLO and MoveNet to be invoked from userspace.

The application was developed in Zig, which is responsible for the event loop, worker thread orchestration, synchronization, and dataflow between modules. C++ bridge layers are integrated via the Zig C interface and compiled into a single self-contained binary, to import libraries not yet available in the Zig ecosystem. OpenCV and GStreamer handle video capture, processing, overlay rendering, and JPEG encoding; Vitis AI and VART handle loading and execution of the DPU-compiled models. Each library-specific component is isolated from the core application; in principle, as long as the interface contract is respected, the system can operate with different underlying libraries.

### B. DPU Model Deployment

The visual inference pipeline uses two models compiled in `.xmodel` format for the target DPU implementation; both are sourced from the AMD Model Zoo. The first model, YOLOv5-nano, locates the subject within the frame and outputs a bounding box used in subsequent stages, operating on a 640×640 letterboxed frame. The second model, MoveNet INT8, takes as input a 192×192 crop extracted from the bounding box and produces 17 keypoints in COCO17 format.

Input preparation is performed before each DPU invocation. The captured BGR frame is resized and letterboxed as the YOLO model expects; the detected region is then cropped and letterboxed before MoveNet invocation. The bounding box is smoothed via Exponential Moving Average between YOLO invocations to reduce jitter on the MoveNet crop. MoveNet outputs are then mapped from the COCO-17 schema to an internal representation retaining only joint placements; for display purposes, coordinates are projected back to the original frame coordinate space.

The DPU pipeline result is stored in a `PoseFrame` containing the bounding box, keypoints, timestamps, and profiling information.

### C. Measurement Setup

Measurements are collected by the `Analyzer` module during execution. For each processed frame, the system records capture time, person detection time, pose estimation time, render submission time, and end-to-end latency. End-to-end

latency is measured from frame availability to motion analysis completion. Every 60 frames, the Analyzer aggregates the collected measurements and prints a runtime summary for monitoring purposes, reporting mean throughput, average latency per stage, loop overruns, and dropped frames. In addition to latency metrics, we collect information on power consumption and temperature.

## V. Experimental Results

### A. Edge-Pipeline Performance

We evaluated the system over ten independent runs of 30 minutes each, with a 5-minute cooldown period between runs. To test the system independently of input variability, each run was performed on the same video: a 90-second squat sequence looped to reach 30 minutes; this avoids contamination of results due to differing inputs and camera limitations. Across the ten runs, the system processed 1,050,295 frames over 17,999.2 s of execution, corresponding to an average of 105,030 $\pm$ 259 frames per session and a mean throughput of 58.35 $\pm$ 0.14 FPS. Table I reports the throughput obtained in each run. Each value corresponds to the average frame rate sustained during one 30-minute execution.

TABLE I
Throughput measured in each 30-minute run.

| Run | FPS | Run | FPS |
|---|---|---|---|
| 1 | 58.207 | 6 | 58.541 |
| 2 | 58.434 | 7 | 58.575 |
| 3 | 58.381 | 8 | 58.220 |
| 4 | 58.461 | 9 | 58.203 |
| 5 | 58.241 | 10 | 58.261 |
| **Mean** | | 58.35 $\pm$ 0.14 | |

Table II summarizes the aggregate measurements collected across the ten runs, including throughput, end-to-end latency, stage-level execution time, energy consumption, and thermal behavior. The mean end-to-end latency was 17.07 $\pm$ 0.05 ms, while the median latency was 17.30 ms. The maximum observed latency for a single run ranged between 66.10 ms and 69.17 ms, due to YOLO invocation. The 99th-percentile latency was 20.71 ms. Although the event loop is capped at 60 Hz, yielding a budget of 16.67 ms per frame, the median latency slightly exceeds this budget, which accounts for the observed loop overruns. Nonetheless, the system sustained a stable frame rate within 3% of the 60 FPS target across all runs. Across all runs, the average energy cost was 0.103 $\pm$ 0.0003 J/frame, equivalent to an average board-level power of 6.03 W. The maximum recorded temperature across sensors was 44.34 °C, within the thermal operating range.

Since LLM feedback is generated on a remote server, its latency depends on both network conditions and the server's token generation speed. For this reason, LLM latency is recorded for completeness but is not used as a performance metric.

### B. Stage-Level Latency Breakdown

The largest share of execution time in the edge pipeline is spent on MoveNet inference. Across runs, it required 13.56 $\pm$ 0.03 ms on average, with a median latency of 14.45 ms. The 99th-percentile latency for the MoveNet stage ranged between 14.45 ms and 17.30 ms depending on the run. Overall, the total pose estimation time matches the MoveNet inference time, averaging 13.57 $\pm$ 0.03 ms, indicating that preprocessing, postprocessing, and motion analysis add only limited overhead.

The capture stage required 1.67 $\pm$ 0.04 ms on average, with a 99th-percentile latency of 2.86 ms, confirming that the single-slot buffer effectively decoupled acquisition from the main event loop, reducing the impact of camera or file input latency on the rest of the pipeline. Rendering required 1.81 $\pm$ 0.03 ms for submission, while the render worker itself required approximately 31.98 $\pm$ 0.10 ms. Having the worker on a separate thread significantly improved real-time pipeline performance. The high render worker times are justified by the nature of its workload: JPEG encoding and overlay generation are compute-intensive and are therefore executed outside the real-time pipeline.

Due to the tuning of `detect_interval` to the lowest possible value, YOLOv5-nano does not contribute significantly to the average cost. Each invocation of YOLO requires approximately 50.1 $\pm$ 0.9 ms, but the detector is configured to run only on the first frame. This decision is justified by our use case: the squat involves limited subject displacement, so once detected, the person is unlikely to exit the bounding box, allowing the cost of detection to be amortized over the entire run.

Based on these results, the primary bottleneck of the system is pose inference, while frame acquisition, control flow, and worker interaction contribute only secondarily to the overall execution cost. The breakdown also confirms the role of the asynchronous design in keeping rendering and communication outside the real-time inference path.

## VI. Conclusion

In this work, we have presented FLEX, an edge-cloud system for real-time exercise coaching on the AMD Kria KV260. The system's design partitions workloads such that latency-sensitive pipeline stages run on the edge device, leveraging available FPGA accelerators to perform pose detection via deep neural networks. Only high-level descriptions are permitted to leave the board. This enables computationally intensive tasks to be offloaded while preserving user privacy, as video frames and keypoint coordinates are not exposed to third-party inference providers.

Experimental results demonstrate that the system can sustain near-60 FPS operation across repeated executions. Across ten independent 30-minute runs, the system processed more than 1.05 million frames, reaching an average throughput of 58.35 $\pm$ 0.14 FPS, a median end-to-end latency of 17.30 ms, a p99 latency of 20.71 ms, and an average energy cost of 0.103 $\pm$ 0.0003 J/frame. Stage-level analysis showed that MoveNet inference is the dominant cost, while capture, rendering, submission, and communication are far from being

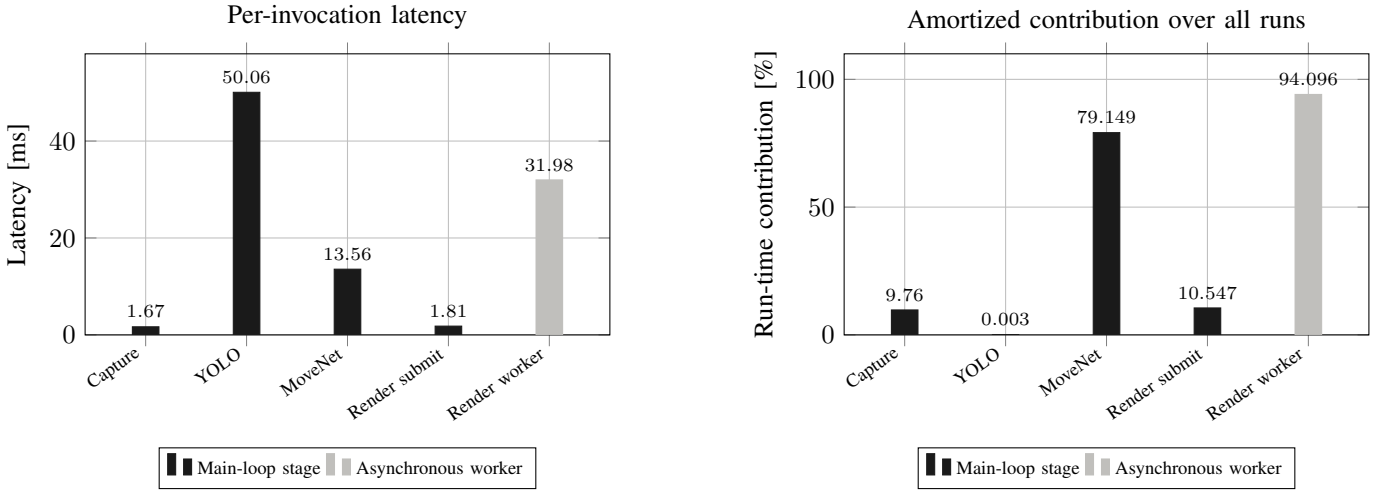



Fig. 3. Stage-level latency and amortized run-time contribution. The left plot reports the average latency of a single invocation of each stage. The right plot reports the cumulative execution time of each stage divided by the total active execution time across the ten runs. YOLOv5-nano has high per-invocation latency, but its amortized contribution is only 0.0028% because it is invoked once per 30-minute run in the evaluated squat sequence. The render worker is highlighted separately because rendering is asynchronous and is not part of the main real-time inference path.

TABLE II
AGGREGATE PERFORMANCE OVER TEN 30-MINUTE RUNS.

| **Metric** | **Value** |
|---|---|
| Run duration | 30 min |
| Cooldown between runs | 5 min |
| Total active runtime | 17,999.2 s |
| Total processed frames | 1,050,295 |
| Frames per run | 105,030 $\pm$ 259 |
| Throughput | 58.35 $\pm$ 0.14 FPS |
| Mean end-to-end latency | 17.07 $\pm$ 0.05 ms |
| Median end-to-end latency | 17.30 ms |
| 99th percentile latency | 20.71 ms |
| Maximum latency | 66.10–69.17 ms |
| Capture latency | 1.67 $\pm$ 0.04 ms |
| YOLOv5-nano latency | 50.1 $\pm$ 0.9 ms |
| MoveNet INT8 latency | 13.56 $\pm$ 0.03 ms |
| Pose-estimation latency | 13.57 $\pm$ 0.03 ms |
| Render-submission latency | 1.81 $\pm$ 0.03 ms |
| Render-worker latency | 31.98 $\pm$ 0.10 ms |
| Energy per frame | 0.103 $\pm$ 0.0003 J/frame |
| Equivalent board power | 6.03 W |
| Maximum temperature | 44.34 °C |
| CPU utilization | 45.44 $\pm$ 3.02% |

bottlenecks due to the asynchronous design. The results indicate that heterogeneous FPGA devices are a suitable target for pose estimation DNN workloads under real-time constraints, and that non-real-time-sensitive components can be offloaded to remote providers while respecting user privacy constraints.

Future work may extend the system with exercises beyond the squat, evaluate the quality of generated feedback against expert-provided references, and develop solutions for dynamic exercises with greater displacement, requiring more frequent updates of the region of interest.

## REFERENCES


[1] Deloitte and EuropeActive, *European Health & Fitness Market Report 2024*. EuropeActive, 2024.

[2] S. Lowe and G. ÓLaighin, "The age of the virtual trainer," *Procedia Engineering*, vol. 34, pp. 242–247, 2012.

[3] X. Jin, Y. Yao, Q. Jiang, X. Huang, J. Zhang, and X. Zhang, "Virtual personal trainer via the Kinect sensor," in *Proc. IEEE 16th Int. Conf. Communication Technology (ICCT)*, Hangzhou, China, Oct. 2015.

[4] S. E. H. Mahmoud and Z. A. E. H. Taha, "AI personal trainer for lateral raises and shoulder presses exercises," in *Proc. 2023 Intelligent Methods, Systems, and Applications (IMSA)*, Giza, Egypt, Jul. 2023.

[5] A. Abosbaa and N. Drawil, "Virtual smart mirror as a personal trainer using machine learning," in *Proc. 2024 IEEE 4th Int. Maghreb Meeting Conf. Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, Tripoli, Libya, May 2024.

[6] Y. Al-Ameri, "FPGA-based hardware accelerators for deep learning in mobile robotics," M.S. thesis, Dept. Computing, Univ. Turku, Turku, Finland, 2023.

[7] S. Aranda Lizano, "Comparison of edge computing platforms for hardware acceleration of AI: Kria KV260, Jetson Nano and RTX 3060," M.S. thesis, Dept. Computing, Univ. Turku, Turku, Finland, 2024.

[8] K. Lata, P. Singh, S. Saini, and L. R. Cenkeramaddi, "Privacy-preserving brain tumor detection using FPGA-accelerated deep learning on Kria KV260 for smart healthcare," *Computer Methods and Programs in Biomedicine Update*, vol. 8, p. 100205, 2025.

[9] S. N. Ramesh, M. Sarkar, M. Audette, and C. Paolini, "Efficient real-time fall prediction and detection using privacy-centric vision-based human pose estimation on the Xilinx Kria K26 SOM," in *Proc. 2023 IEEE Biomedical Circuits and Systems Conf. (BioCAS)*, Toronto, ON, Canada, Oct. 2023.

[10] B. Feng, D. Zhang, H. Zhang, C. Wang, and X. Zheng, "Qwen2.5-0.5B inference optimization based on FPGA acceleration," in *Proc. 2025 4th Int. Conf. Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIoTC)*, 2025.

[11] M. Xiang, R. Fernando, and B. Wang, "On-device Qwen2.5: Efficient LLM inference with model compression and hardware acceleration," *arXiv preprint arXiv:2504.17376*, 2025.

[12] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 591–624, First Quarter 2023.

[13] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[14] Y. He, J. Fang, F. R. Yu, and V. C. Leung, "Large language models (LLMs) inference offloading and resource allocation in cloud-edge computing: An active inference approach," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11253–11264, Dec. 2024.
[15] Google DeepMind, "Gemma 3 1B IT model card," Hugging Face, 2025. [Online]. Available: https://huggingface.co/google/gemma-3-1b-it. Accessed: Jun. 8, 2026.
[16] Gemma Team, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.