Assistive AI Mobility in Healthcare: Comparative Evaluation of Single Board Computers with AI/GPU Support for Real-Time Object Detection

Nurdaulet Asset, Martin Obert and Stefan Twieg
Department of Electrical, Mechanical and Industrial Engineering, Anhalt University of Applied Sciences, Bernburger Str. 55, 06366 Köthen, Germany
Nurdaulet.Asset@student.hs-anhalt.de, Martin.Obert@hs-anhalt.de, Stefan.Twieg@hs-anhalt.de

Keywords: NVIDIA Jetson Orin Nano, NVIDIA Jetson Nano, Raspberry Pi 5, YOLO, TensorRT, Hailo AI HAT+, Embedded AI, Computer Vision, Assistive Mobility, Rollator, Benchmarking, Energy Efficiency, Cost-Performance.

Abstract: Artificial intelligence (AI) and computer vision enable modern assistive systems. Within the AktiMuW project, we present a multi-sensor rollator designed under strict latency, energy, and cost constraints to assess posture and provide guidance for safe, comfortable walking. We evaluate three single-board computers (SBCs): NVIDIA Jetson Nano, NVIDIA Jetson Orin Nano, and Raspberry Pi 5 with AI HAT+, by running the same YOLO-based detector on an identical rollator-perspective video, using each platform's native acceleration (TensorRT on Jetson, Hailo runtime on Raspberry Pi 5) and PyTorch as a baseline where applicable. We report accuracy, end-to-end latency and inference time, frames per second (FPS), CPU/GPU utilization, energy per inference, and cost per FPS. Results show clear trade-offs: TensorRT on Jetson Orin Nano delivers the highest throughput and lowest latency; Raspberry Pi 5 with Hailo offers the best energy efficiency and cost per FPS; Jetson Nano achieves moderate real-time performance at low device cost. These findings provide practical guidance for selecting embedded AI hardware in assistive mobility, healthcare robotics, and smart environments, and they inform the design of an energy-aware, cost-effective prototype for the AktiMuW rollator.

1 INTRODUCTION

1.1 Motivation

The aging population in Germany and across Europe is steadily increasing [1], [2], creating urgent demand for technological solutions that can support older adults in maintaining independence and safety. Artificial intelligence (AI) and computer vision have become crucial enablers for modern assistive technologies [3], especially in mobility aids[4]-[7]. Within the AktiMuW project, we develop a multi-sensor rollator designed to help elderly people remain active and safe by combining posture monitoring, obstacle detection, and real-time guidance. One of the key features of this system is the use of a stereo-camera module and AI-based object detection to prevent accidents and provide feedback to the user [8].

1.2 Problem

While deep-learning models such as YOLO offer high accuracy for object detection [9], deploying them on embedded platforms introduces challenges. Single board computers (SBCs) like the Raspberry Pi 5, NVIDIA Jetson Nano, and NVIDIA Jetson Orin Nano are attractive due to their small form factor and energy efficiency, but they face strict limitations in latency, throughput (FPS), and power consumption. Selecting the optimal hardware requires careful benchmarking, since excessive delay or high energy demand would reduce usability in real-time assistive robotics.

1.3 Goal

The goal of this study is to systematically compare three representative SBCs - NVIDIA Jetson Nano,

NVIDIA Jetson Orin Nano, and Raspberry Pi 5 with AI HAT+ - when running the same YOLO-based detector on an identical rollator-perspective dataset [10]-[13]. By analyzing key metrics such as accuracy, latency, FPS, CPU utilization, energy per inference, and cost-to-performance, we aim to identify the most suitable hardware platform for integration into the AktiMuW rollator prototype [14].

2 METHODS

2.1 Dataset

The evaluation dataset consists of a single rollator-perspective video file (Rawdata.mp4), (Fig. 1). It captures realistic walking scenarios, including sidewalks, vehicles, pedestrians, and obstacles. This dataset was used uniformly across all platforms to ensure fair comparison.

Figure 1: Example frame from Rawdata.mp4.

2.2 Models

We evaluated YOLO-based models in three runtime formats, all derived from the same baseline Ultralytics PyTorch checkpoint (640.pt) with input size fixed at 640×640 and the segmentation task enabled:

- PyTorch (.pt) - the standard model executed with the Ultralytics library [9]. No conversion was required; the .pt file was used directly for inference on both GPU (CUDA, FP16) and CPU.
- TensorRT (.engine) - an optimized version of the model built with TensorRT 10.3 on Jetson Orin Nano. The export pipeline followed Ultralytics' workflow:
  a) Export .pt → .onnx [15].
  b) Convert .onnx → .engine using TensorRT with FP16 kernels enabled [11], [16]. The resulting .engine file runs only on NVIDIA Jetson devices, where it leverages GPU acceleration to achieve significantly lower latency and higher FPS compared to the .pt runtime.
- Hailo (.hef) - a compiled accelerator model executed on the Hailo-8L NPU mounted on the Raspberry Pi 5 via the AI HAT+. The workflow uses the Hailo Dataflow Compiler (DFC) to transform an ONNX model into a .hef (Hailo Executable File) [13], [17], [18]. This format can run only on Raspberry Pi with the AI HAT+, where inference is fully offloaded to the NPU while the CPU handles video decoding and overlay.

This export pipeline is summarized as:

- .engine for TensorRT on NVIDIA Jetson.
- .hef for Hailo on Raspberry Pi.

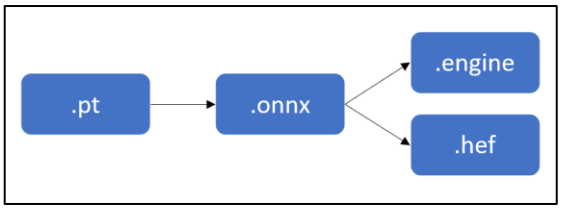

Figure 2: Conversion of the model.

Figure 2 illustrates the process of converting a PyTorch model (.pt) into an ONNX intermediate format, which can then be compiled into either a TensorRT engine (.engine) for NVIDIA Jetson or a Hailo executable (.hef) for Raspberry Pi with AI HAT+ [15]-[17].

2.3 Hardware Platforms

To evaluate performance, three single-board computers (SBCs) were selected. Each platform was chosen to represent a different balance of performance, efficiency, and cost.

The Jetson Orin Nano is the latest high-performance SBC from NVIDIA, featuring Ampere GPU cores and TensorRT 10.3 acceleration (Fig. 3). It provides strong parallel processing capabilities and dedicated AI hardware, enabling efficient inference at higher FPS with reduced latency. Orin Nano was tested both with the baseline PyTorch model (.pt) and the optimized TensorRT engine (.engine) [10], [11].

The Jetson Nano represents NVIDIA's cost-efficient entry-level device, widely used for education, prototyping, and embedded AI [11], (Fig. 4). It features Maxwell GPU cores and supports

TensorRT 8.x acceleration, although performance is limited compared to Orin Nano. For this work, Nano was benchmarked using a TensorRT-optimized .engine file, providing a baseline for low-power inference.

Figure 3: NVIDIA Jetson Orin Nano board.

Figure 4: NVIDIA Jetson Nano board.

The Raspberry Pi 5 is a general-purpose SBC that, when combined with the Hailo AI HAT+, gains access to the Hailo-8L NPU accelerator (13 TOPS), (Fig. 5). In this configuration, the Pi handles preprocessing, video decoding, and overlay, while the Hailo-8L NPU executes inference using the compiled .hef file. This setup provides competitive FPS-per-Watt performance and demonstrates the potential of low-cost accelerator-based solutions for real-time AI applications [12], [13], [18].

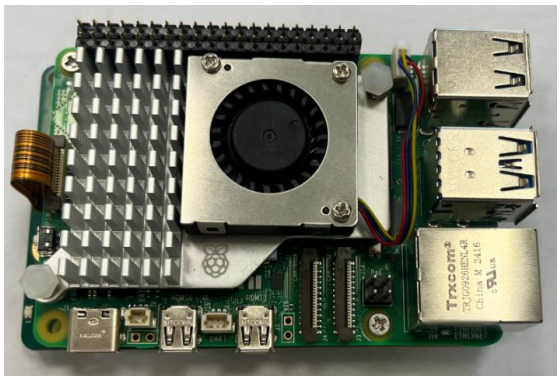

Figure 5: Raspberry Pi 5 with Hailo AI HAT+.

2.4 Software Environment

All three hardware platforms were configured with software environments suitable for benchmarking YOLO-based inference.

NVIDIA Jetson Orin Nano:

- Operating System: Ubuntu 22.04 LTS (JetPack 6.0, L4T r36.4);
- Python: 3.10 (venv);
- Frameworks: PyTorch 2.8.0, TorchVision 0.23, Ultralytics YOLO;
- Acceleration: TensorRT 10.3 (FP16), CUDA 12.6, cuDNN 9.3;
- Tools: OpenCV, psutil, tegrastats [19].

NVIDIA Jetson Nano:

- Operating System: Ubuntu 18.04 LTS (JetPack 4.6, L4T r32.7);
- Python: 3.6 (venv);
- Frameworks: legacy PyTorch build, Ultralytics YOLO;
- Acceleration: TensorRT 8.2.1 (FP16), PyCUDA;
- Tools: OpenCV 4.1.1, psutil, tegrastats [20].

Raspberry Pi 5 with Hailo AI HAT+:

- Operating System: Raspberry Pi OS 64-bit (Bookworm);
- Python: 3.11 (venv);
- Frameworks: Ultralytics YOLO (CPU), HailoRT runtime with GStreamer plugins [17];
- Tools: OpenCV, numpy, psutil.

In all setups, the software stack was chosen to maximize compatibility with each device's hardware capabilities. The Jetson platforms benefited from CUDA and TensorRT for GPU acceleration, while the Raspberry Pi 5 relied on the Hailo-8L NPU to offload inference and reduce CPU load. This ensured that each platform was evaluated under its optimal runtime configuration.

2.5 Metrics

The following metrics were collected during benchmarking of YOLO-based models on the three hardware platforms:

- Accuracy (%) - detection accuracy on the labeled dataset, reported as the proportion of correctly identified objects.
- End-to-End Latency (ms) - the average time per frame, including video decoding, preprocessing, model inference, postprocessing, and overlay.
- Inference Latency (ms) - the model-only forward-pass time, excluding preprocessing and postprocessing.
- Frames Per Second (FPS) - throughput of the video pipeline, measured as the number of frames processed per second.
- CPU Utilization (%) - average process CPU usage during inference, as reported by psutil.

In addition, power and cost-related efficiency metrics were derived using the following formulas:

- Power (W): measured average device power consumption during inference (via tegrastats or external wattmeter).
- Energy per Inference (J):

$$\text{Energy per interface(J)} = \frac{\text{Power(W)}}{\text{FPS}} \quad (1)$$

This represents the joules consumed to process a single video frame.

- FPS per Watt:

$$\text{FPS per Watt} = \frac{\text{FPS}}{\text{Power(W)}} \quad (2)$$

This shows how many frames can be processed per second for each watt of power consumed.

- Cost per FPS (€):

$$\text{Cost per FPS} = \frac{\text{Device Price(€)}}{\text{FPS}} \quad (3)$$

This normalizes device cost by achieved throughput, providing a measure of cost efficiency.

3 RESULTS

To evaluate inference performance across edge platforms, each runtime was executed on the identical rollator-perspective video (Rawdata.mp4) with the same model configuration (YOLO-Seg, 640×640). We first present qualitative visual examples (Figures 6-11) to show that detection/segmentation quality is consistent across runtimes, followed by quantitative Tables 1-2 reporting latency, FPS, CPU load, power-normalized efficiency, and cost-normalized performance.

3.1 Visual Examples (Qualitative)

Figures 6-11 present representative frames from the same rollator-perspective video (Rawdata.mp4) processed with identical YOLO settings (640×640) to visually compare the outputs across platforms.

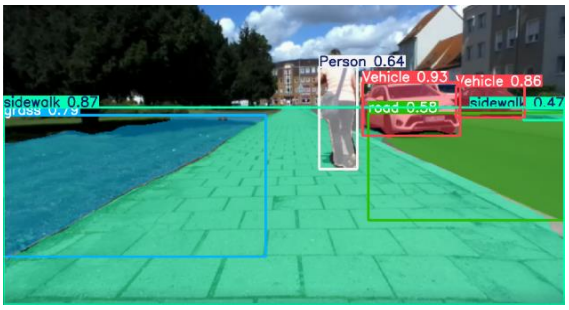

Figure 6: Jetson Orin Nano - TensorRT .engine (FP16, GPU).

Figure 6 describes a representative frame showing high-throughput inference and smooth overlay; this configuration delivers the best latency/FPS on Orin Nano.

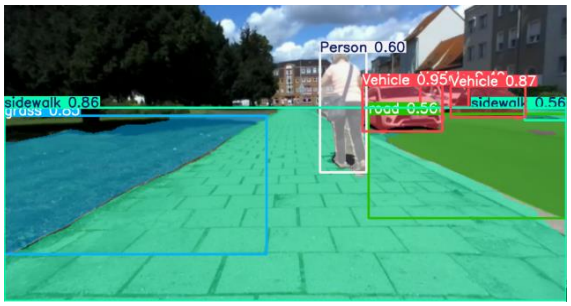

Figure 7: Jetson Orin Nano - PyTorch .pt (GPU).

Figure 7 describes a representative frame with accurate detection and segmentation results, but lower throughput compared to TensorRT on Orin Nano.

Figure 8 describes a representative frame that demonstrates identical visual accuracy but much lower FPS and higher latency, as inference is executed on CPU only.

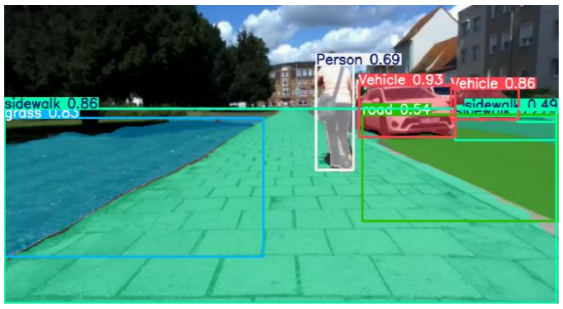

Figure 8: Jetson Orin Nano - PyTorch .pt (CPU-only).

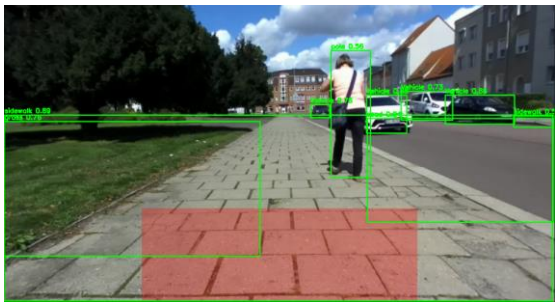

Figure 9: Jetson Nano - TensorRT. engine (FP16, GPU).

Figure 9 describes a representative frame showing moderate throughput and accurate overlays, making Nano a cost-efficient platform for prototyping.

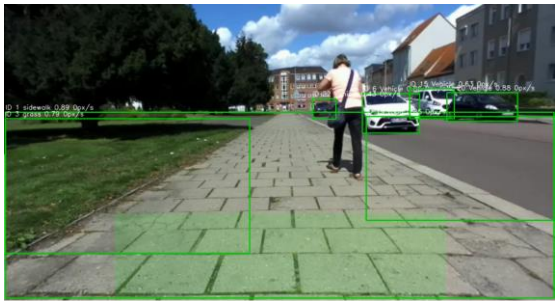

Figure 10: Raspberry Pi 5 - PyTorch .pt (CPU).

Figure 11: Raspberry Pi 5 - Hailo .hef (GPU).

Figure 10 describes a representative frame with correct detection but significantly reduced FPS due to CPU bottlenecks on the Pi 5.

Figure 11 describes a representative frame showing high-throughput inference offloaded to the Hailo-8L accelerator, resulting in stable overlays with low CPU utilization. Figures 9-11 show bounding-box detection results without segmentation masks. This is because the Hailo runtime (and some TensorRT exports) only supported YOLO in detection mode, not segmentation, at the same time of testing. In contrast, Figures 6-8 show the complete segmentation outputs produced by the PyTorch and TensorRT implementations on Jetson Orin Nano, where each detected object is outlined with a segmentation mask rather than only a bounding box.

3.2 Quantitative Results (Tables)

Following the qualitative examples, we report quantitative performance metrics across the three devices. All results were obtained from the Rawdata.mp4 rollator dataset, processed with segmentation at 640×640 resolution. Each experiment was run headless (no GUI display) to ensure reproducibility of FPS and latency numbers.

Power-related metrics (Energy per inference, FPS per Watt) were derived from measured or assumed average device power. Cost per FPS was calculated using typical market prices (€300 for Orin Nano, €120 for Nano, €80/€160 for Pi 5 CPU/Hailo).

Table 1 summarizes results for Jetson Orin Nano across three runtimes: TensorRT (.engine, GPU), PyTorch (.pt, GPU), and PyTorch (.pt, CPU-only).

Table 2 reports results for Jetson Nano (TensorRT) and Raspberry Pi 5 (PyTorch CPU and Hailo NPU). Frame counts differ because each run was limited by dataset duration and pipeline speed.

3.3 Results Interpretation

The quantitative results highlight clear trade-offs across devices and runtimes.

On Jetson Orin Nano, TensorRT optimization provides the best overall performance, reaching nearly 39 FPS with an end-to-end latency of ~31 ms. Compared to the PyTorch GPU run 20 FPS, TensorRT delivers almost 2× higher throughput and significantly lower inference latency. The CPU-only configuration, by contrast, is not practical for real-time use, dropping to ~2 FPS and showing very high latency >400 ms.

Table 1: NVIDIA Jetson Orin Nano (headless, FP16, 640×640).

Device	NVIDIA Jetson Orin Nano	NVIDIA Jetson Orin Nano	NVIDIA Jetson Orin Nano
Model	TensorRT .engine 16FP	PyTorch .pt	PyTorch .pt
GPU/CPU	GPU	GPU	CPU
Frames	873	873	873
FPS	38.81	20.11	2.361
e2e ms	31.44	53.51	418.41
Infer ms	7.89	32.84	403.71
CPU%	102.69	109.00	229.77
Power (W)	7.389	7.125	6.485
Energy/inf (J)	0.190	0.354	2.747
FPS/W	5.253	2.822	0.364
Cost/FPS	7.73	14.92	127.09
Accuracy (%)	94	94	94

Table 2: Jetson Nano and Raspberry Pi 5 (headless, FP16, 640×640).

Device	NVIDIA Jetson Nano	Raspberry Pi 5	Raspberry Pi 5
Model	TensorRT .engine 16FP	PyTorch .pt	Hailo .hef
GPU/ CPU	GPU	CPU	GPU
Frames	873	436	1738
FPS	11.32	7.75	27.31
e2e ms	90.46	126.36	36.39
Infer ms	63.62	112.72	-
CPU%	64.21	368.18	39.07
Power (W)	6.50	6.50	8.00
Energy/inf (J)	0.57	0.82	0.29
FPS/W	1.74	1.19	3.41
Cost/ FPS	10.60	10.32	5.86
Accuracy (%)	93	92	85

On Jetson Nano, TensorRT acceleration also proves essential. The Nano achieves ~11 FPS with ~90 ms latency - far slower than Orin Nano but still capable of lightweight real-time inference for simpler tasks. Without GPU acceleration, Nano would not be suitable for segmentation workloads.

On Raspberry Pi 5, the gap between general-purpose CPU inference and accelerator-based inference is most pronounced. The PyTorch CPU run achieves ~8 FPS with >120 ms latency, while the Hailo NPU reaches 27 FPS with only ~36 ms latency. This makes the Pi 5 + Hailo AI HAT+ a compelling low-power, cost-efficient solution, despite slightly lower accuracy 85%.

Detection-only runs (e.g., Hailo .hef) reach higher FPS due to reduced post-processing but provide less visual detail. Segmentation runs (Jetson Orin Nano TensorRT/PyTorch) offer richer spatial information but at slightly higher latency. Object detection accuracy across all formats remained similar (Jetson Orin Nano TensorRT/PyTorch ~94%, Jetson Nano TensorRT ~93%, Pi 5 Hailo ~85%, Pi 5 PyTorch ~92%), indicating that segmentation mode mainly influences visualization and processing load, not class-level accuracy.

To better illustrate these results, Figures 12 and 13 show bar plots of GPU performance and CPU performance across the tested platforms.

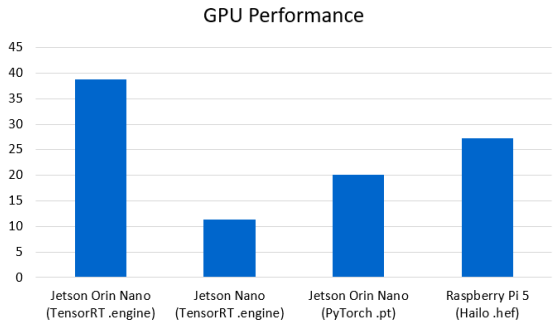

Figure 12: GPU performance.

Figure 12 compares GPU-accelerated runtimes, highlighting the clear advantage of TensorRT on Orin Nano and Hailo on Raspberry Pi 5.

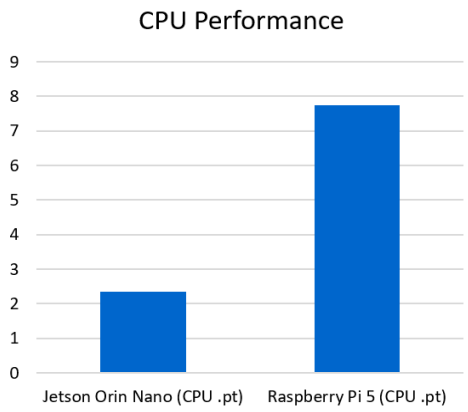

Figure 13: CPU performance.

Figure 13 focuses on CPU inference, where both Orin Nano and Pi 5 demonstrate significantly lower throughput compared to their GPU counterparts, confirming that CPU-only execution is not suitable for real-time segmentation.

3.4 Practical Implications for AktiMuW

From a cost-performance perspective, Jetson Orin Nano with TensorRT is the most capable platform. It combines the highest accuracy 94% with real-time throughput ~39 FPS and low latency ~31 ms, making it ideal for development, testing, and deployment in demanding robotics and healthcare scenarios. For the AktiMuW rollator, this means reliable perception, faster reaction to hazards, and headroom for adding additional sensors or AI tasks without overloading the system.

At the same time, Raspberry Pi 5 with Hailo AI HAT+ demonstrates the best energy efficiency and cost-to-performance ratio. With ~27 FPS at only 8 W power draw, this configuration is attractive for battery-powered rollators or low-budget healthcare devices, where long operating time and affordable components are critical. Although accuracy is slightly lower 85%, the trade-off may be acceptable for real-world deployment where affordability and portability are prioritized.

Jetson Nano represents a middle ground. With ~11 FPS at modest cost, it offers a viable entry-level solution for prototyping or educational versions of the rollator. However, its limited throughput and higher latency compared to Orin Nano or Hailo mean that it may struggle with complex or safety-critical real-time applications.

In summary, for the AktiMuW project:

- Best performance: Jetson Orin Nano with TensorRT.
- Best efficiency and affordability: Raspberry Pi 5 with Hailo AI HAT+.
- Moderate, low-cost baseline: Jetson Nano.

This comparison provides actionable guidance: Orin Nano is recommended for development and high-performance trials, while Pi 5 + Hailo is best suited for scalable, cost-sensitive deployments of the assistive rollator platform.

4 CONCLUSIONS

This study compared three single-board computing platforms - NVIDIA Jetson Orin Nano, NVIDIA Jetson Nano, and Raspberry Pi 5 with Hailo AI HAT+ - for real-time object detection and segmentation in the context of the AktiMuW smart rollator project. Using the same YOLO-based model and a standardized dataset, we evaluated accuracy, latency, throughput, CPU utilization, energy efficiency, and cost-to-performance.

The results highlight clear trade-offs. Jetson Orin Nano with TensorRT delivers the best overall performance, achieving nearly 39 FPS with low latency and high accuracy, making it the most suitable choice for demanding robotics and healthcare applications. Raspberry Pi 5 with Hailo excels in energy efficiency and cost-effectiveness, offering ~27 FPS at low power draw, which is highly attractive for battery-powered or budget-constrained deployments. Jetson Nano, while capable of ~11 FPS, serves best as a low-cost prototyping platform but lacks the performance or efficiency required for large-scale deployment.

For the AktiMuW rollator, these findings suggest two clear options: Jetson Orin Nano is recommended for the best-performing version of the system, while Raspberry Pi 5 with Hailo AI HAT+ is recommended for the cost-efficient version. By systematically benchmarking across devices, this work provides practical guidance for selecting embedded AI hardware in assistive mobility systems and contributes to broader discussions on deploying AI for healthcare and aging societies.

5 ACKNOWLEDGMENTS

The research project underlying this contribution was funded by the Federal Ministry of Education and Research under the grant number FKZ 03WIR3122A.

The responsibility for the content of this publication lies with the author.

REFERENCES

[1] World Health Organization, "Ageing and Health," Oct. 2025, [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/ageing-and-health, [Accessed: Feb. 24, 2026].

[2] Eurostat, Ageing Europe: Looking at the Lives of Older People - 2023 Edition. Luxembourg: Publications Office of the European Union, 2023, [Online]. Available: https://ec.europa.eu/eurostat.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436-444, May 2015, doi: 10.1038/nature14539.

[4] E. J. Harris, I. Khoo, and E. Demircan, "A survey of human gait-based artificial intelligence applications," Frontiers in Robotics and AI, vol. 8, p. 749274, 2021, doi: 10.3389/frobt.2021.749274.

[5] A. Macie, T. Matson, and A. Schinkel-Ivy, "Age affects the relationships between kinematics and postural stability during gait," Gait & Posture, vol. 102, pp. 86-92, 2023, doi: 10.1016/j.gaitpost.2023.03.004.

[6] E. Blasch et al., "Machine learning/artificial intelligence for sensor data fusion - Opportunities and challenges," IEEE Aerospace and Electronic Systems Magazine, vol. 36, no. 7, pp. 80-93, Jul. 2021, doi: 10.1109/MAES.2020.3049030.

[7] S. M. Alfayeed and B. S. Saini, "Human gait analysis using machine learning: A review," in Proc. International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, UAE, pp. 550-554, 2021, doi: 10.1109/ICCIKE51210.2021.9410678.

[8] S. Twieg, R. V. Menghani, S. Rajanayagam, and V. Kurumbaparambil, "Sensor-based gait analysis: A comparative study of ultrasonic and laser sensors for gait monitoring in rollator-assisted walking," 2025, [Online]. Available: https://www.researchgate.net/publication/392553747_Sensor-Based_Gait_Analysis_A_Comparative_Study_of_Ultrasonic_and_Laser_Sensors_for_Gait_Monitoring_in_Rollator-Assisted_Walking, [Accessed: Feb. 24, 2026].

[9] Ultralytics, "YOLO Docs - Models and Export (ONNX, TensorRT, etc.)," [Online]. Available: https://docs.ultralytics.com/, [Accessed: Feb. 24, 2026].

[10] NVIDIA, "Jetson Orin Nano Developer Kit," [Online]. Available: https://developer.nvidia.com/embedded/jetson-orin-nano-developer-kit, [Accessed: Feb. 24, 2026].

[11] NVIDIA, "Jetson Linux Documentation," [Online]. Available: https://docs.nvidia.com/jetson/, [Accessed: Feb. 24, 2026].

[12] Raspberry Pi, "Raspberry Pi 5 - Product Page & Documentation," [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/, [Accessed: Feb. 24, 2026].

[13] Raspberry Pi, "AI HAT+ - Documentation," [Online]. Available: https://www.raspberrypi.com/documentation/accessories/ai-hat-plus.html, [Accessed: Feb. 24, 2026].

[14] R. V. Menghani, S. Rajanayagam, and S. Twieg, "Optimized indoor and outdoor SLAM for a quadruped-based robot walking partner to safely promote elderly mobility and support nursing staff," in Proc. Irish Signals and Systems Conference (ISSC 2025), Letterkenny, Ireland, Jun. 9-10, 2025.

[15] ONNX, "Open Neural Network Exchange (ONNX) - Specification and Tooling," [Online]. Available: https://onnx.ai/, [Accessed: Feb. 24, 2026].

[16] NVIDIA, "TensorRT Documentation," [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/, [Accessed: Feb. 24, 2026].

[17] Hailo, "AI Software Suite (HailoRT, Dataflow Compiler, Plugins)," [Online]. Available: https://hailo.ai/products/hailo-software/hailo-ai-software-suite/, [Accessed: Feb. 24, 2026].

[18] Hailo, "Hailo-8 / Hailo-8L Product Brief and Datasheets," 2023, [Online]. Available: https://hailo.ai/products/hailo-8/, [Accessed: Feb. 24, 2026].

[19] NVIDIA, "tegrastats - System Monitoring Utility," Jetson Linux Developer Guide, Jan. 16, 2026, [Online]. Available: https://docs.nvidia.com/jetson/, [Accessed: Feb. 24, 2026].

[20] NVIDIA, "nvpmodel and jetson_clocks - Power and Performance Controls," Jetson Linux Developer Guide, [Online]. Available: https://docs.nvidia.com/jetson/, [Accessed: Feb. 24, 2026].