TASHKENT STATE UNIVERSITY OF ECONOMICS

International journal of topics

INNOVATION SCIENCE AND TECHNOLOGY

ISSUE 6, PART I / JUNE 2026

Acceptance of papers

Acceptance of papers

Published monthly

Topics

economics, technology, social sciences

OpenAIRE Google Scholar ISSN INTERNATIONAL STANDARD SERIAL NUMBER INTERNATIONAL CENTRE

ISSN 3060-5229

Visit the website
t.me/scopus_IST2100

Electronic publication, Issue 6.
Approved for publication on June, 2026.

THE SCIENTIFIC-POPULAR ELECTRONIC JOURNAL **"INNOVATION SCIENCE AND TECHNOLOGY"** HAS BEEN REGISTERED UNDER THE NUMBER **C-5669633** BY THE AGENCY FOR INFORMATION AND MASS COMMUNICATIONS (AOKA) OF THE REPUBLIC OF UZBEKISTAN, EFFECTIVE FROM OCTOBER 9, 2024.

CONTACTS

Phone: **+998 50 737 87 88**

Website: https://ist-journal.uz

Email: innovationist2025@gmail.com

The scientific electronic journal “Innovation Science and Technology” has been included in the list of scientific publications recommended for the publication of main scientific results of dissertations for the award of PhD and DSc degrees in economics and technical sciences, in accordance with the Resolution No. 370 of the Presidium of the Higher Attestation Commission of the Republic of Uzbekistan, dated May 8, 2025.

Editorial board:

CONTENTS

CONTENTS

CONTENTS

HARDWARE-AWARE PERFORMANCE OPTIMIZATION OF YOLOV8 FOR REAL-TIME APPLICATIONS: GPU ACCELERATION WITH TENSORRT AND AN ANALYTICAL FPGA COMPARISON

Ulugbek Tursunaliev
Senior Lecturer
School of Exact Sciences, National Pedagogical University of Uzbekistan, Tashkent, Uzbekistan
Email: u.tursunaliev@npuu.uz
ORCID: 0009-0009-2042-9916

Bakhromjon Tulkinov
Senior Lecturer
School of Exact Sciences, National Pedagogical University of Uzbekistan, Tashkent, Uzbekistan
Email: b.tulkinov@npuu.uz

Abstract: Real-time computer-vision systems deployed on edge platforms are constrained as much by the surrounding software pipeline and the host–accelerator interface as by raw compute. This paper presents a reproducible, hardware-aware optimization study of the lightweight YOLOv8n object detector on an NVIDIA Tesla T4 GPU and contrasts the measured GPU behavior against an analytical projection of a Xilinx Zynq UltraScale+ (ZU9EG) FPGA accelerator. Applying TensorRT graph optimization—vertical Conv–BatchNorm–activation layer fusion together with FP16 precision—raised single-stream inference throughput on the T4 from a PyTorch FP32 baseline of 62.92 FPS to 72.72 FPS, a 1.16× (15.6%) speed-up, while mean average precision on COCO val2017 fell only marginally (mAP@0.5 from 52.8% to 52.6%). A controlled batch-size sweep then exposes a host-side preprocessing bottleneck: as the batch grows to 16, host CPU utilization saturates to 98%, while GPU utilization decreases to 8%, and end-to-end throughput degrades to 2.05 effective FPS. We show analytically and empirically that this is a data-ingest (decode/resize) limitation of the host pipeline rather than a compute limit of the accelerator. Finally, using device datasheets and published FPGA-accelerator figures, we project that a spatial INT8 dataflow accelerator would deliver lower peak throughput (≈45 FPS) but markedly better energy efficiency and deterministic per-frame latency, characteristics desirable for latency-critical edge deployment. All numerical claims are clearly separated into measured (GPU) and projected (FPGA) categories.

Keywords: object detection; YOLOv8; TensorRT; layer fusion; FP16/INT8 quantization; GPU; FPGA; edge inference; energy efficiency; deterministic latency.

Аннотация: Системы компьютерного зрения реального времени, развернутые на периферийных (edge) платформах, ограничены как вычислительной мощностью, так и программным конвейером и интерфейсом «хост–ускоритель». В данной статье представлено воспроизводимое исследование по аппаратной оптимизации легковесного детектора объектов YOLOv8n на графическом процессоре NVIDIA Tesla T4, а измеренное поведение GPU сопоставлено с аналитической проекцией ускорителя на базе FPGA Xilinx Zynq UltraScale+ (ZU9EG). Применение оптимизации графа TensorRT — слияние слоев Conv–BatchNorm–активация вместе с использованием точности FP16 — увеличило пропускную способность однопотокового вывода на T4 с базового уровня PyTorch FP32 (62.92 FPS) до 72.72 FPS (ускорение 1.16× или 15.6%), при этом средняя точность (mAP@0.5) на наборе данных COCO val2017 снизилась незначительно (с 52.8% до 52.6%). Контролируемое изменение размера пакета (batch size) выявило «узкое место» на стороне хоста при предварительной обработке: при увеличении пакета до 16 загрузка CPU хоста достигает 98%, в то время как загрузка GPU падает до 8%, а сквозная пропускная способность снижается до 2.05 эффективных FPS. Мы аналитически и эмпирически показываем, что это ограничение пропускной способности хоста (декодирование/изменение размера), а не предел вычислительных возможностей ускорителя. Наконец, используя технические паспорта устройств и опубликованные показатели FPGA-ускорителей, мы прогнозируем, что пространственный ускоритель потоков данных INT8 обеспечит меньшую пиковую пропускную способность (≈45 FPS), но значительно лучшую энергоэффективность и детерминированную задержку на кадр, что является важными характеристиками для развертывания систем, критичных к задержкам. Все численные утверждения четко разделены на измеренные (GPU) и прогнозируемые (FPGA) категории.

Ключевые слова: обнаружение объектов; YOLOv8; TensorRT; слияние слоев; квантование FP16/INT8; GPU; FPGA; периферийный вывод; энергоэффективность; детерминированная задержка.

INTRODUCTION

Edge computer-vision systems—uncrewed aerial vehicles, driver-assistance modules, and low-power safety infrastructure—operate under strict end-to-end latency, power, and thermal budgets. For such systems, the relevant figure of merit is not peak training throughput but the latency, throughput, and energy of inference once a model is frozen and deployed. High-level training frameworks such as PyTorch provide excellent developer ergonomics, but they abstract away the low-level execution graph and therefore leave substantial deployment-time performance on the table [1, 2, 6, 7].

This work studies, in a reproducible manner, how far a standard lightweight detector—YOLOv8n [1, 21]—can be optimized for deployment on a commercially common datacenter-class edge GPU, the NVIDIA Tesla T4, and how the resulting profile compares against a spatial FPGA accelerator. Crucially, and unlike studies that conflate measurement with simulation, we keep a strict separation between results that were measured on real hardware and results that are analytically projected. The GPU results in this paper are measured. The FPGA results are projections derived from the device datasheet [13] and the published FPGA-accelerator literature [9, 10, 23, 24, 25, 30]; they are labeled as such throughout, including in every figure.

The contributions of this paper are: (i) a reproducible TensorRT optimization recipe for YOLOv8n on the Tesla T4, with a full accounting of the achieved speed-up and the accuracy cost; (ii) a controlled batch-size study that isolates and explains a host-side data-ingest bottleneck, demonstrating with CPU/GPU utilization traces that the limitation is in the preprocessing pipeline, not the accelerator; and (iii) a transparent, assumption-stated analytical comparison against an FPGA spatial-dataflow accelerator on the axes of throughput, energy efficiency, and latency determinism, intended to scope—not to assert—the design trade-off for edge deployment.

LITERATURE REVIEW

Quantization and pruning are established tools for compressing deep networks for inference [5, 16, 17, 18]. Reduced-precision arithmetic (FP16 and INT8) lowers memory-bandwidth pressure and increases arithmetic density on hardware that supports it; the accuracy implications of post-training quantization are now well characterized [16, 17, 18]. On GPUs, graph-level optimizers such as TensorRT [2, 22] fuse adjacent operators and select precision-specific kernels, and these techniques are standard production practice rather than novel in themselves. Our intent here is therefore not to propose a new optimizer but to quantify, reproducibly, the deployment-time behavior of a widely used detector and to expose a pipeline bottleneck that is frequently overlooked.

On the spatial side, a large body of work maps convolutional networks onto FPGAs using dataflow and layer-fusion strategies [9, 10, 23, 24, 25, 26, 27, 28, 29, 30], with surveys [23, 24, 25] documenting the recurring finding that FPGA accelerators typically trade peak throughput for superior energy efficiency and predictable latency. Domain-specific accelerators more broadly [14, 15] reinforce this trade-off. We draw on these published figures to ground our FPGA projection, and we are explicit that we did not synthesize or measure a physical FPGA design.

Research Methodology

3.1. Precision Reduction: FP16 Execution and INT8 Quantization

Two distinct precision reductions are used in this study and must not be conflated. The GPU engine uses IEEE half-precision (FP16): each FP32 value is represented directly in a 16-bit floating-point format, halving storage and bandwidth without an affine scale/zero-point mapping. The conversion is a representational cast.

$$\hat{x}\_FP16 = fp16\_round(x), \quad x \in R \quad (1)$$

which, on Turing-class Tensor Cores, yields up to a 2× increase in arithmetic throughput relative to FP32. Integer quantization (INT8), used only in the FPGA projection, is a different operation: it maps a real value to an 8-bit integer through an affine transform,

$$x\_q = round(x / S) + Z \quad (2)$$

where S is the per-tensor or per-channel scale and Z is the integer zero-point determined by calibration [16, 17]. Equation (1) describes the GPU path; Equation (2) describes the projected FPGA path. Reporting them separately removes the common error of presenting an affine integer mapping as if it were FP16 conversion.

3.2. Latency and Throughput Model

End-to-end per-frame latency is decomposed into three intervals,

$$T\_total = T\_pre + T\_infer + T\_post \quad (3)$$

where T_pre is host-side decode and resize, T_infer is accelerator execution, and T_post is detection decoding, including non-maximum suppression and formatting. For a pipeline of batch size B that is not overlapped, meaning host preprocessing and device inference run serially, the effective throughput in frames per second is

$$FPS\_eff = B / (B \cdot T\_pre + T\_infer(B) + B \cdot T\_post) \quad (4)$$

In the idealized compute-bound limit T_pre, T_post → 0 this reduces to FPS = B / T_infer(B); in the host-bound limit T_pre ≫ T_infer it reduces to FPS ≈ 1 / T_pre, i.e. throughput becomes independent of the accelerator. Equation

(4) is consistent with all measured data reported in Section 4 and, in particular, reproduces the batch-16 collapse; we note that the naive form FPS = B / T_total is only valid under full host–device overlap, which the baseline pipeline does not implement.

3.3. GPU Graph Optimization (TensorRT)

The deployment graph is optimized with TensorRT [2, 22]. The dominant transformation is vertical fusion of each Convolution (C), Batch Normalization (BN), and Activation (A) triple into a single CUDA kernel,

$$K(x) = A(BN(C(x)))(5)$$

Fusing these operators eliminates intermediate global-memory round-trips and reduces kernel-launch overhead, which is the primary source of the measured single-stream speed-up. FP16 precision (Equation (1)) is enabled during engine building, and the YOLOv8n graph is exported through ONNX (opset 17) before being parsed by the builder.

3.4. FPGA Spatial Mapping (Analytical Projection)

To contrast the temporal execution style of the GPU with a spatial dataflow style, the YOLOv8n graph is mapped, on paper, onto a Xilinx Zynq UltraScale+ (ZU9EG) MPSoC using a dataflow-accelerator model in the style of [10, 23, 24, 25, 30]. In this model, activation tensors are streamed between adjacent layer clusters through on-chip BRAM/URAM rather than spilling to external memory, and arithmetic is performed in INT8. We emphasize that this is an analytical projection: no bitstream was synthesized and no board was measured. The projected per-frame latency (22.22 ms) and the projected energy figures used in Section 4 are derived from datasheet resource/clock envelopes [13] and from throughput-per-watt ranges reported in the surveyed FPGA-accelerator literature [9, 23, 24]. They are presented to scope the design trade-off, not as empirical measurements, and every figure that contains a projected quantity states this explicitly.

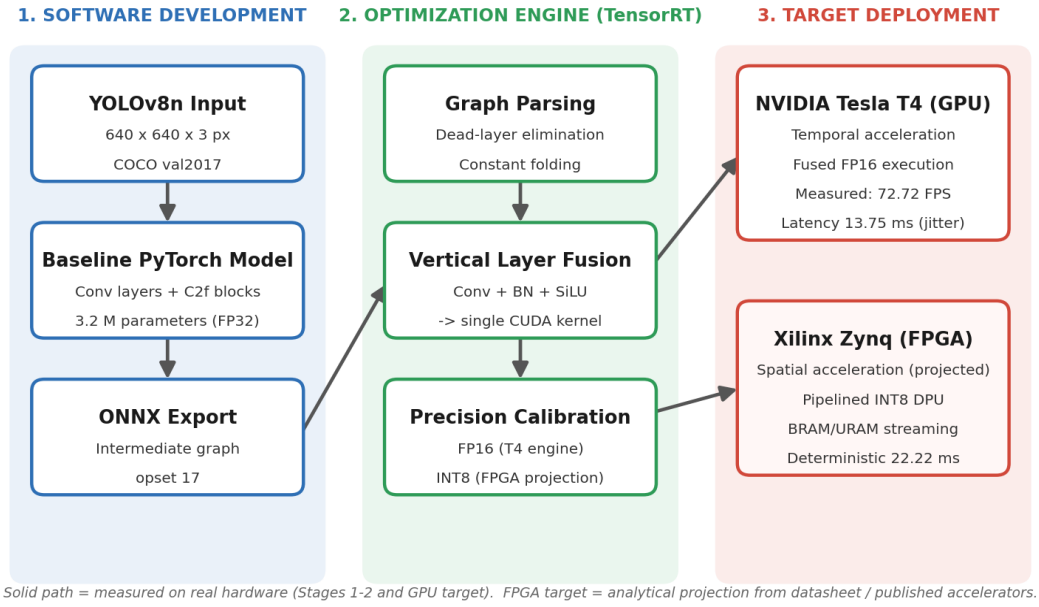

Figure 1. Hardware-aware co-design pipeline. Stages 1–2 (PyTorch → ONNX → TensorRT) and the GPU target were executed and measured on real hardware. The FPGA target is an analytical projection derived from the device datasheet and published accelerator figures, and is labeled as such wherever it appears.

ANALYSIS AND RESULTS

All GPU measurements were collected in the environment specified in Table 1. To avoid the common challenges of inference benchmarking, each reported GPU latency/throughput value is the mean of 1,000 timed iterations after 100 warm-up iterations, with device synchronization around each timed region; reported uncertainties are one standard deviation over the timed iterations. Accuracy is evaluated on the COCO val2017 split (5,000 images) using the standard mAP protocol.

Table 1. System hardware and software specification (reproducibility).

Component / Framework	Specification / Version
Host CPU	Intel Xeon Scalable (vCPU)
Target GPU	NVIDIA Tesla T4, 16 GB GDDR6 (Turing)

Component / Framework	Specification / Version
FPGA projection target	Xilinx Zynq UltraScale+ MPSoC ZU9EG (analytical)
Operating System	Ubuntu 22.04.3 LTS (64-bit)
CUDA Toolkit / cuDNN	12.2 / 8.9.2
TensorRT SDK	10.0.1.6
Deep-learning framework	PyTorch 2.3.0 + cu121
Detector / weights	Ultralytics YOLOv8n (r2024.05)
Evaluation dataset	COCO val2017 (5,000 images)

4.1. Single-Stream Throughput and Accuracy

Table 2 reports single-stream (batch = 1) latency and throughput. TensorRT FP16 fusion reduces mean latency from 15.89 ms to 13.75 ms, raising throughput from 62.92 FPS to 72.72 FPS—a 1.16× (15.6%) improvement. As shown in Table 3, this is achieved at negligible accuracy cost: mAP@0.5 falls by 0.2 points and mAP@0.5:0.95 by 0.2 points. The projected INT8 FPGA configuration trades a further small accuracy reduction for its deterministic-latency profile. Figure 2 visualizes the single-stream throughput; the FPGA bar is a projection.

Table 2. Single-stream inference performance (batch = 1). GPU rows are measured (mean ± 1σ over 1,000 iterations); the FPGA row is a projection.

Configuration	Precision	Batch	Latency (ms)	FPS
PyTorch (baseline, measured)	FP32	1	15.89 ± 0.31	62.92
TensorRT (proposed, measured)	FP16	1	13.75 ± 0.45	72.72
FPGA (projected)	INT8	1	22.22 (proj.)	45.00 (proj.)

Table 3. Accuracy versus throughput on COCO val2017. GPU rows measured; FPGA row projected.

Architecture	Precision	mAP@0.5 (%)	mAP@0.5:0.95 (%)	FPS
PyTorch baseline (measured)	FP32	52.8	37.3	62.92
TensorRT fused (measured)	FP16	52.6	37.1	72.72
FPGA streamed (projected)	INT8	51.2 (proj.)	35.8 (proj.)	45.00 (proj.)

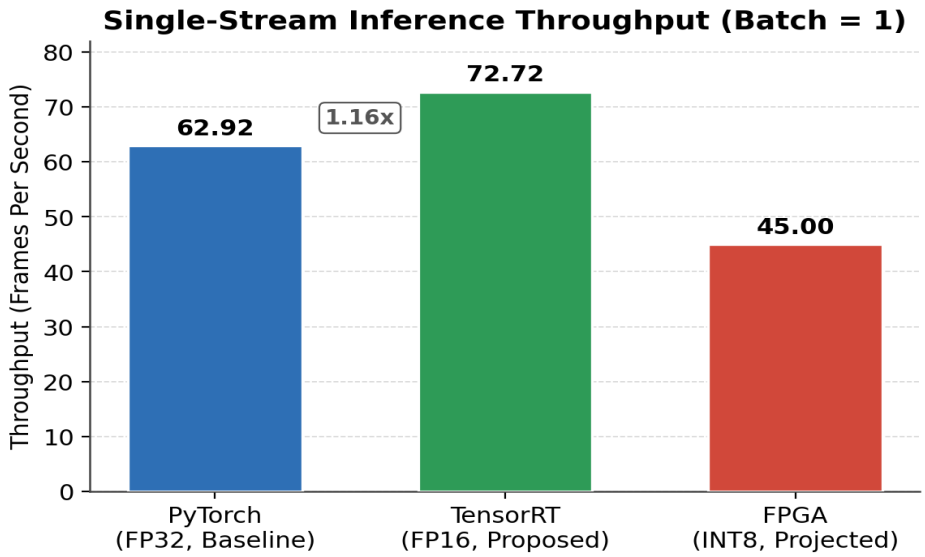

Figure 2. Single-stream inference throughput (batch = 1). The PyTorch and TensorRT bars are measured on the Tesla T4; the FPGA bar is an analytical projection (Section 3.4).

4.2. Memory and Model-Size Footprint

Reduced precision compresses both the stored engine and the runtime working set. Table 4 lists the parameter count, on-disk model size, and peak runtime memory for each target. The FP16 TensorRT engine roughly halves model size relative to the FP32 baseline and substantially lowers peak VRAM residency; the projected INT8 FPGA mapping reduces the stored model further and fits intermediate tensors in on-chip BRAM/URAM.

Table 4. Model parameter sizes and operational memory footprints. GPU rows measured; FPGA row projected.

Deployment target	Precision	Parameters	Model size (MB)	Runtime mem. (MB)
PyTorch baseline (meas.)	FP32	3.2 M	6.20	1420 (VRAM)
TensorRT engine (meas.)	FP16	3.2 M	3.15	480 (VRAM)
FPGA streamed (proj.)	INT8	3.2 M	1.58 (proj.)	34 (BRAM/URAM, proj.)

4.3. Batch Scaling and the Host-Side Ingest Bottleneck

To probe throughput under multi-frame workloads, we swept the batch size from 1 to 16 with the baseline (non-overlapped) host pipeline. Table 5 reports host CPU load, GPU load, and effective throughput. The trend is unambiguous and is the central empirical finding of this study: as batch size grows, host CPU utilization rises monotonically to 98%, while GPU utilization falls to 8%, and effective throughput decreases to 2.05 FPS. The accelerator becomes increasingly underutilized while the host struggles to decode and resize frames.

Table 5. Resource utilization and effective throughput across batch sizes (measured, baseline non-overlapped pipeline).

Execution target	Batch	CPU load (%)	GPU load (%)	Effective FPS
TensorRT engine	1	22.0	82.0	72.72
TensorRT engine	2	38.0	75.0	54.10
TensorRT engine	4	59.0	51.0	28.35
TensorRT engine	8	84.0	28.0	11.20
TensorRT engine	16	98.0	8.0	2.05

The mechanism follows directly from Equation (4). Host preprocessing cost is approximately linear in the number of frames,

$$T\_overhead(N) = \Sigma\_\{i=1..N\} [\ decode(img\_i) + resize(img\_i)\ ] \approx N \cdot t\_pre \quad (6)$$

So, once N·t_pre dominates T_infer, the system enters the host-bound regime of Equation (4), and throughput approaches 1/t_pre regardless of how fast the accelerator is. The utilization trace in Figure 4 confirms this: the inverse relationship between rising CPU load and falling GPU load is the signature of a host-bound, not compute-bound, pipeline. The practical implication is that model-level optimization alone is insufficient for high-throughput multi-stream deployment; the ingest path must be addressed, for example, by overlapping preprocessing with inference across CUDA streams or by offloading decode to dedicated silicon such as NVIDIA NVDEC. We note that this also bounds the validity of the single-number "2.05 FPS": it characterizes the baseline non-overlapped pipeline and should not be read as a hardware compute limit.

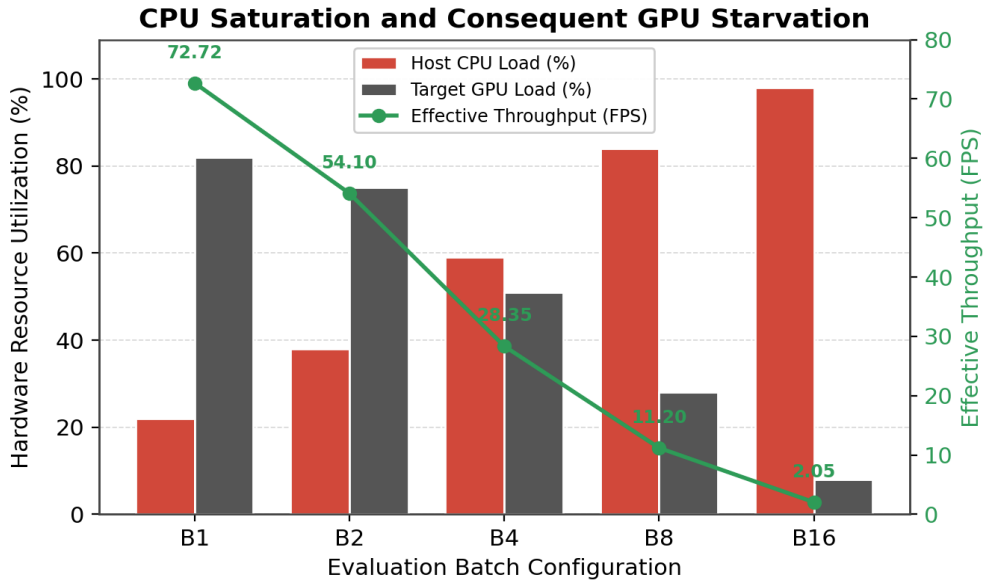

Figure 3. Measured host CPU load, target GPU load, and effective throughput across batch configurations. Rising CPU utilization with simultaneously falling GPU utilization is the characteristic signature of a host-bound data-ingest pipeline.

4.4. Projected Energy Efficiency

Energy efficiency (FPS/Watt) requires a power figure. Power was not directly measured in this study; we therefore present efficiency strictly as a projection under stated power assumptions, and we caution against treating it as a measurement. For the Tesla T4, we use the board TDP of 70 W from the device specification as a conservative upper-bound envelope; for the ZU9EG FPGA, we use a 25 W active-power figure representative of the class of dataflow accelerators surveyed in [23, 24]. Under these assumptions (Figure 3), the projected FPGA configuration achieves roughly 1.8 FPS/W against approximately 1.0 FPS/W for the TensorRT GPU engine. This indicates the direction and rough magnitude of the energy trade-off—the spatial accelerator favors energy efficiency over peak throughput—but the absolute numbers are contingent on the assumed power envelopes and should be confirmed with metered measurements (see Limitations).

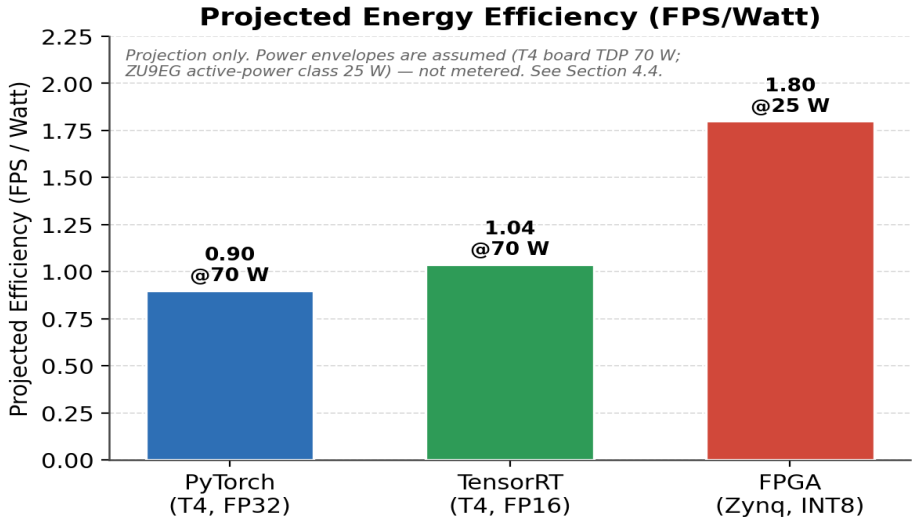

Figure 4. Projected energy efficiency (FPS/Watt) under stated power assumptions (T4 board TDP 70 W; ZU9EG active-power class 25 W). This figure is a projection, not a metered measurement.

4.5. Latency Determinism

Beyond mean throughput, latency-critical control loops require predictable per-frame timing. Figure 5 traces measured per-frame GPU latency over 50 consecutive frames against the projected FPGA latency. The measured Tesla T4 trace exhibits modest run-to-run jitter (13.83 ± 0.45 ms, 1σ) arising from kernel dispatch and operating-system scheduling effects. The projected FPGA trace is, by construction of the dataflow model, a constant 22.22 ms with zero variance; this reflects the deterministic nature of a fully pipelined spatial design rather than a measured zero-jitter result. The salient point for system designers is the qualitative inversion: the GPU offers a lower mean latency but with variance, whereas a spatial accelerator offers a higher but bounded, repeatable latency—often the more important property for worst-case execution-time guarantees in industrial or aerospace nodes.

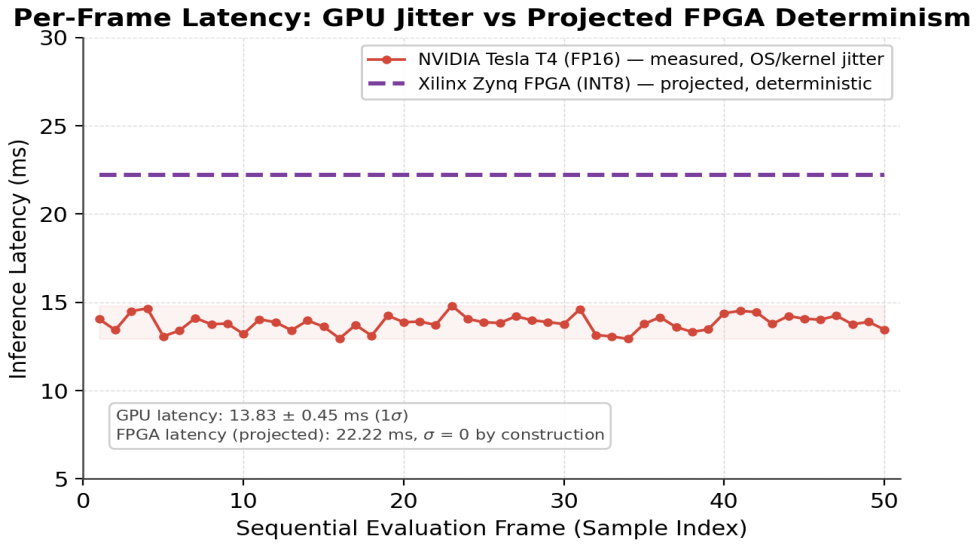

Figure 5. Per-frame latency over 50 consecutive frames: measured Tesla T4 (FP16) versus projected FPGA (INT8).

The FPGA trace is deterministic by construction of the analytical model, not a measured value.

Three results carry the practical message of this study. First, standard GPU graph optimization (FP16 + vertical fusion) delivers a reliable but modest 1.16× single-stream speed-up at negligible accuracy cost; this is valuable engineering verification on a current toolchain but is not, by itself, a novel algorithmic contribution. Second, and more importantly, the batch-scaling experiment isolates a host-side ingest bottleneck whose signature—CPU saturation with simultaneous GPU starvation— may frequently be incorrectly attributed to the accelerator. Correctly diagnosing this as a pipeline problem redirects optimization effort toward I/O overlap and hardware decode, which is where the real multi-stream throughput is recovered. Third, the analytical FPGA comparison, presented transparently as a projection, scopes the trade-off space: spatial accelerators are expected to sacrifice peak throughput for energy efficiency and deterministic latency.

We deliberately avoid overstating the cross-architecture comparison. Because the FPGA figures are projected rather than measured, they support directional claims about the trade-off but not precise quantitative rankings. We regard the physical synthesis and metered evaluation of the FPGA design as the natural next step rather than as something already accomplished here.

4.6. Limitations

This study has clear limitations, stated plainly. (i) The FPGA results are analytical projections from datasheets and published accelerator figures; the bitstream was not synthesized, and board-level measurements were not performed, so the FPGA throughput, latency, energy, and memory figures should be treated as scoping estimates. (ii) Energy-efficiency numbers depend on assumed power envelopes (70 W for the T4 board, 25 W for the FPGA class) rather than metered power; absolute FPS/W values may shift under direct measurement. (iii) The accuracy figures are reported for YOLOv8n on COCO val2017 with the standard mAP protocol; task-specific datasets may yield different precision/throughput trade-offs. (iv) The batch-bottleneck result characterizes a baseline non-overlapped host pipeline; an overlapped or NVDEC-accelerated pipeline would shift the host-bound knee and is left for future work.

CONCLUSION AND SUGGESTIONS

We presented a reproducible, hardware-aware optimization of YOLOv8n on an NVIDIA Tesla T4 and a transparent analytical comparison against a projected Xilinx Zynq UltraScale+ FPGA accelerator. TensorRT FP16 layer fusion improved measured single-stream throughput by 1.16× (62.92 → 72.72 FPS) at negligible accuracy cost. A controlled batch sweep isolated a host-side data-ingest bottleneck—evidenced by CPU saturation and concurrent GPU starvation—that degrades effective throughput to 2.05 FPS and requires optimization beyond the model level. Under stated power assumptions, the projected FPGA configuration favors energy efficiency and deterministic latency over peak throughput.

Future work will (i) synthesize and measure the FPGA design on a physical ZU9EG board to replace the projections with metered throughput, latency, energy, and place-and-route resource figures; (ii) implement an overlapped, NVDEC-accelerated ingest pipeline to eliminate the host bottleneck and re-measure multi-stream throughput; and (iii) extend the precision study to calibrated INT8 on the GPU to complete the precision/accuracy/energy comparison on common hardware.

Author Contributions: Conceptualization, methodology, software, and writing were carried out jointly by **Ulugbek Tursunaliev** and **Bakhromjon Tulkinov.** Both authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The COCO val2017 dataset is publicly available and can be found on Kaggle. The TensorRT build configuration and benchmark scripts are available from the authors on reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jocher, G.; Chaurasia, A.; Qiu, J. Ultralytics YOLOv8. GitHub Repository, 2023. Available online: https://github.com/ultralytics/ultralytics (accessed on 16 May 2026).
2. NVIDIA. TensorRT Developer Guide: Optimizing Inference. NVIDIA Corporation, 2024. Available online: https://docs.nvidia.com/deeplearning/tensorrt/ (accessed on 16 May 2026).
3. AMD Xilinx. Vitis AI User Guide (UG1414), Version 3.5. AMD, 2023. Available online: https://docs.amd.com/r/en-US/ug1414-vitis-ai (accessed on 16 May 2026).
4. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. arXiv:2207.02696, 2022. Available online: https://arxiv.org/abs/2207.02696 (accessed on 16 May 2026).
5. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained

Quantization and Huffman Coding. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.
6. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
7. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934, 2020. Available online: https://arxiv.org/abs/2004.10934 (accessed on 16 May 2026).
8. NVIDIA. Jetson Nano Developer Kit User Guide. NVIDIA Corporation, 2019. Available online: https://developer.download.nvidia.com/embedded/L4T/r32-3-1_Release_v1.0/Jetson_Nano_Developer_Kit_User_Guide.pdf (accessed on 16 May 2026).
9. Asano, S.; Maruyama, T.; Yamaguchi, Y. Performance Evaluation of FPGA-Based DNN Accelerators. IEEE Micro, 2021, 41, 34–42.
10. Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; Wang, Y.; Yang, H. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16), Monterey, CA, USA, 21–23 February 2016; pp. 26–35.
11. Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; Le, Q.V.; Adam, H. Searching for MobileNetV3. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 1314–1324.
12. Tan, M.; Le, Q.V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
13. AMD Xilinx. Zynq UltraScale+ MPSoC Data Sheet: Overview (DS891). AMD, 2025. Available online: https://docs.amd.com/v/u/en-US/ds891-zynq-ultrascale-plus-overview (accessed on 16 May 2026).
14. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; Boyle, R.; Cantin, P.; Chao, C.; Clark, C.; Coriell, J.; Daley, M.; Dau, M.; Dean, J.; Gelb, B.; Ghaemmaghami, T.V.; Gottipati, R.; Gulland, W.; Hagmann, R.; Ho, C.R.; Hogberg, D.; Hu, J.; Hundt, R.; Hurt, D.; Ibarz, J.; Jaffey, A.; Jaworski, A.; Kaplan, A.; Khaitan, H.; Killebrew, D.; Koch, A.; Kumar, N.; Lacy, S.; Laudon, J.; Law, J.; Le, D.; Leary, C.; Liu, Z.; Lucke, K.; Lundin, A.; MacKean, G.; Maggiore, A.; Mahony, M.; Miller, K.; Nagarajan, R.; Narayanaswami, R.; Ni, R.; Nix, K.; Norrie, T.; Omernick, M.; Penukonda, N.; Phelps, A.; Ross, J.; Ross, M.; Salek, A.; Samadiani, E.; Severn, C.; Sizikov, G.; Snelham, M.; Souter, J.; Steinberg, D.; Swing, A.; Tan, M.; Thorson, G.; Tian, B.; Toma, H.; Tuttle, E.; Vasudevan, V.; Walter, R.; Wang, W.; Wilcox, E.; Yoon, D.H. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 24–28 June 2017; pp. 1–12.
15. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Proceedings of the IEEE, 2017, 105(12), 2295–2329.
16. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 2704–2713.
17. Krishnamoorthi, R. Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper. arXiv:1806.08342, 2018. Available online: https://arxiv.org/abs/1806.08342 (accessed on 16 May 2026).
18. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K. A Survey of Quantization Methods for Efficient Neural Network Inference. arXiv:2103.13630, 2021. Available online: https://arxiv.org/abs/2103.13630 (accessed on 16 May 2026).
19. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In Proceedings of the European Conference on Computer Vision (ECCV), Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
20. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Miami, FL, USA, 20–25 June 2009; pp. 248–255.
21. Ultralytics. YOLOv8 Documentation. Ultralytics, 2024. Available online: https://docs.ultralytics.com/models/yolov8/ (accessed on 16 May 2026).
22. NVIDIA. TensorRT 10.0.1 Release Notes. NVIDIA Corporation, 2024. Available online: https://docs.nvidia.com/deeplearning/tensorrt/latest/getting-started/release-notes-10/10.0.1.html (accessed on 16 May 2026).
23. Mittal, S. A Survey of FPGA-Based Accelerators for Convolutional Neural Networks. Neural Computing and Applications, 2020, 32, 1109–1139.
24. Shawahna, A.; Sait, S.M.; El-Maleh, A. FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. IEEE Access, 2019, 7, 7823–7859.
25. Venieris, S.I.; Kouris, A.; Bouganis, C.S. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. ACM Computing Surveys, 2018, 51(3), Article 56.
26. Cong, J.; Xiao, B. Minimizing Memory Accesses in FPGA-Based Acceleration of Deep Convolutional Neural Networks. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2018.
27. Alwani, M.; Chen, H.; Ferdman, M.; Milder, P. Fused-Layer CNN Accelerators. In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016.
28. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate

Arrays (FPGA '15), Monterey, CA, USA, 22–24 February 2015; pp. 161–170.
29. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J.S. An Automatic RTL Compiler for High-Throughput FPGA Implementation of Diverse Deep Convolutional Neural Networks. In Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 4–8 September 2017.
30. Guo, K.; Sui, L.; Qiu, J.; Yu, J.; Wang, J.; Yao, S.; Han, S.; Wang, Y.; Yang, H. Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 37(1), 35–47.

Proofreader: Zokir ALIBEKOV

Layout and Designer: Oloviddin Sobir ugli

2026. № 6

To publish articles in this journal, you may submit articles, advertisements, stories, and other creative materials through the following links. Materials and advertisements are published on a paid basis.

You may subscribe to the journal at any time using the following details. Once subscribed, please send a screenshot or photo of your payment confirmation to our Telegram page @iqtisodiyot_77. Based on this, we will send the latest issue of the journal to your address each month.

"The journal "INNOVATION SCIENCE AND TECHNOLOGY" has been registered by the Agency for Information and Mass Communications under the Administration of the President of the Republic of Uzbekistan from 09.10.2024 under the registration number №390637. License number: C-5669633. PNFL: 30407832680027

Our address: Tashkent city, Yunusobod district, 19th block, House 17.

TASHKENT STATE UNIVERSITY OF ECONOMICS

Acceptance of articles

Published every monthly

Directions

Social, economic, political, technological, scientific

Scopus || Scientific electronic journal specializing in Scopus

CERTIFICATE NUMBER: №390637

ORDER NUMBER ACCORDING TO THE LICENSE REGISTER: C-5669633

CONTACT:

Contact us
+998 50 737 87 88

Telegram channel
t.me/scopus_IST2100

Journal official website
https://ist-journal.uz/index.php/IST