ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

Comprehensive Power and Performance Characterization of Deep Neural Networks on Jetson Orin Nano with PyTorch and TensorRT

Tang Zequn

School of Physics and Electronic Science, Hunan University of Science and Technology, Xiangtan, Hunan, 411201, China

Abstract: *Deep learning inference on edge devices must balance speed, energy, memory, temperature, and accuracy. This paper evaluates PyTorch FP32, TensorRT FP32, and TensorRT FP16 inference on an NVIDIA Jetson Orin Nano 8GB using ResNet-18, MobileNetV2, and YOLOv5s. We measure throughput, power, energy, FPS/W, RAM usage, and temperature rise with a 1000 images x 5 repeated workload, while accuracy is evaluated on ImageNet and COCO2017. Results show that TensorRT greatly reduces energy and improves efficiency. ResNet-18 energy decreases from 557.9 J to 113.9 J under TensorRT FP16 with nearly unchanged Top-1 accuracy. MobileNetV2 achieves strong energy reduction but shows a clear FP16 accuracy drop. YOLOv5s TensorRT results are reported as engine-level measurements because the preprocessing and post-processing scope differs from the PyTorch pipeline. The results show that edge deployment should be evaluated as a system-level trade-off rather than by FPS alone.*

Keywords: *Edge AI, Jetson Orin Nano, TensorRT, PyTorch, FP16 inference, power consumption, energy efficiency, memory usage, thermal behavior*

1. Introduction

Deep learning models are increasingly deployed on edge devices such as cameras, robots, and industrial monitoring systems. Compared with cloud inference, edge inference reduces latency, bandwidth use, and privacy risks, but it is limited by power, thermal, memory, and compute resources. Therefore, practical evaluation must consider not only latency or FPS, but also power, energy efficiency, memory footprint, temperature, and accuracy.

NVIDIA Jetson platforms are widely used embedded GPU systems for edge AI. TensorRT can improve inference through graph optimization, layer fusion, kernel tuning, and reduced-precision execution. However, its benefit depends on model architecture and precision mode. FP16 may reduce power and energy, but it does not always improve throughput or preserve accuracy.

This paper characterizes PyTorch and TensorRT inference on Jetson Orin Nano 8GB using ResNet-18, MobileNetV2, and YOLOv5s. The evaluation compares PyTorch FP32, TensorRT FP32, and TensorRT FP16. It reports performance, power, energy, FPS/W, RAM usage, temperature rise, and accuracy, providing a multi-metric view of edge inference behavior.

The main contributions are:

- A systematic benchmark of three models across three inference configurations on Jetson Orin Nano.
- Joint analysis of throughput, power, energy, and energy efficiency (FPS/W).
- Accuracy evaluation on ImageNet and COCO to quantify the accuracy–efficiency trade-off.
- Additional hardware-level insights into RAM usage and temperature variation.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the platform and methodology. Section 4 presents results. Section 5 discusses limitations and deployment implications. Section 6 concludes the paper.

This structure helps separate measurement scope from deployment interpretation. In particular, the study treats throughput, energy, memory, and temperature as coupled system variables, so a configuration is considered practical only when it improves efficiency without introducing

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

unacceptable accuracy loss or resource pressure.

2. Related Work

2.1. Edge AI Benchmarking

Edge AI benchmarking has become important because embedded platforms must meet strict latency, power, and memory constraints. Prior benchmarks, including MLPerf-related and edge-specific studies, show that throughput alone is insufficient for deployment decisions. Recent work also emphasizes energy, accuracy, and reproducibility [1]-[3]. This study follows that direction and further includes RAM and temperature analysis on Jetson Orin Nano.

Compared with recent on-device AI and edge benchmark studies, this work keeps the evaluation scope intentionally focused but system-oriented. Rather than reporting a single latency or FPS number, the benchmark links throughput with energy, RAM usage, temperature rise, and validation accuracy. This makes the results more useful for deployment decisions on resource-limited devices, where a configuration with the highest FPS may not be the best choice if it consumes more power, increases thermal accumulation, or reduces model accuracy.

2.2. TensorRT Acceleration on Embedded GPUs

TensorRT is an NVIDIA inference optimizer that supports layer fusion, kernel selection, and reduced-precision execution [5]. On Jetson devices, it can greatly improve speed and energy efficiency, especially for convolutional networks and object detectors. However, existing work often focuses on FPS, while accuracy changes, memory usage, and thermal behavior are less frequently analyzed together.

2.3. Hardware-Aware Edge Deployment

Hardware-aware deployment considers latency, power, memory, temperature, and accuracy jointly. These factors can change with model type, batch size, precision, and power mode. Our work adopts this system-level view by comparing three models and three inference configurations with aligned telemetry and validation results.

3. Experimental Setup and Methodology

3.1. Hardware Platform

Experiments are conducted on an NVIDIA Jetson Orin Nano 8GB developer kit [4]. The device uses shared LPDDR5 memory and an embedded GPU. All measurements use maximum performance mode (MAXN, nvpmodel -m 0) with jetson_clocks enabled. The ambient temperature is approximately 25 degrees C. tegrastats samples VDD_IN power, RAM, SWAP, and temperature every 500 ms. RAM values are reported as system RAM because of the unified memory architecture.

The hardware and software state is kept consistent across all model configurations. The maximum performance mode and locked clocks reduce variation caused by background power-management changes, while the same ambient condition is maintained for all tests. This control is important because edge devices have small thermal margins, and even modest differences in initial temperature or power mode can change the measured energy and throughput.

3.2. Models and Inference Configurations

We evaluate ResNet-18, MobileNetV2, and YOLOv5s [7]-[9]. Each model is tested with PyTorch FP32, TensorRT FP32, and TensorRT FP16.

For ResNet-18 and MobileNetV2, all configurations use the same batch-size-1 image pipeline with 224 x 224 ImageNet inputs. PyTorch uses the original models [6], while TensorRT uses converted engines.

For YOLOv5s, PyTorch FP32 is measured with the standard Python detection pipeline at 640 x 640 input resolution, including image loading, preprocessing, inference, decoding, and NMS. TensorRT

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

FP32/FP16 performance and power are measured at engine level using the converted engines.

Therefore, YOLOv5s TensorRT values are marked with an asterisk. They indicate engine-level acceleration potential and should not be read as strict end-to-end comparisons with the PyTorch pipeline.

3.3. Batch Size and Input Resolution

All inference experiments use batch size 1, which reflects latency-oriented edge scenarios where images usually arrive sequentially.

ResNet-18 and MobileNetV2 use 1 x 3 x 224 x 224 inputs. YOLOv5s uses 1 x 3 x 640 x 640 inputs, which partly explains its higher power, memory, and energy values. The input settings are summarized in Table 1.

Table 1: Batch Size and Input Resolution.

Model	Task	Batch Size	Input Tensor Shape
ResNet-18	Classification	1	1 x 3 x 224 x 224
MobileNetV2	Classification	1	1 x 3 x 224 x 224
YOLOv5s	Detection	1	1 x 3 x 640 x 640

3.4. Datasets and Workload

Performance and power are measured using 1000 images repeated five times, giving 5000 single-image inferences per configuration. Accuracy is evaluated separately on ImageNet validation data for classification and COCO2017 validation data for detection.

3.5. Measurement Protocol

Each experiment uses a 180 s window with idle-before, inference, and idle-after phases. tegrastats samples VDD_IN power and telemetry every 500 ms. A wrapper adds monotonic timestamps to tegrastats lines, while the inference program records start and end times using the same CLOCK_MONOTONIC clock.

The idle-before and idle-after periods also provide a visual check on the separation between background power and active inference power. In post-processing, the inference interval is treated as the only valid region for energy calculation. This avoids mixing idle samples into the workload energy estimate and makes the comparison between slow PyTorch runs and shorter TensorRT runs more consistent.

Only samples within the inference interval are used. Boundary power values are linearly interpolated, and energy is computed by trapezoidal integration. This alignment reduces timing error and avoids wall-clock drift.

Using the same monotonic clock for the inference program and telemetry logger further reduces alignment error. Although the sampling interval cannot capture every short transient spike, timestamp alignment and interpolation provide a reproducible workload-level estimate. The resulting energy values should therefore be interpreted as integrated inference energy over the complete 5000-run workload, not as kernel-level instantaneous power.

All configurations are processed with the same logging and post-processing scripts. This reduces bias from manual window selection and makes the reported energy, FPS/W, RAM usage, and temperature rise comparable across models, while still preserving the distinction between end-to-end classification runs and YOLOv5s TensorRT engine-level measurements.

Total inference energy is calculated by integrating the aligned power samples over the inference interval:

$$E = \int_{t_s}^{t_e} P\,(t)\,dt \approx \sum_{i=1}^{N-1} \frac{P_i + P_{i+1}}{2}(t_{i+1} - t_i).$$

Energy efficiency is reported as FPS/W:

FPS/W = Throughput (FPS) / Average Power (W).

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

3.6. YOLOv5s TensorRT Measurement Scope

YOLOv5s uses different measurement scopes for PyTorch and TensorRT. The PyTorch FP32 result is end-to-end and includes loading, resizing, tensor preparation, network execution, decoding, and NMS. TensorRT FP32/FP16 results mainly characterize exported engine execution and exclude parts of the Python pipeline.

This choice isolates TensorRT engine behavior on Jetson Orin Nano, but it limits direct cross-framework comparison. Therefore, YOLOv5s TensorRT values are treated as engine-level results in all tables and figures.

The 1000 x 5 workload is reported as a stable aggregate measurement. Future work will retain per-repeat timestamps to report standard deviation and range for FPS, power, energy, FPS/W, memory, and temperature.

3.7. Experimental Workflow

Figure 1 shows the workflow. Models are evaluated under each configuration while tegrastats logs telemetry using the same monotonic time base. Accuracy is evaluated separately on the corresponding validation datasets.

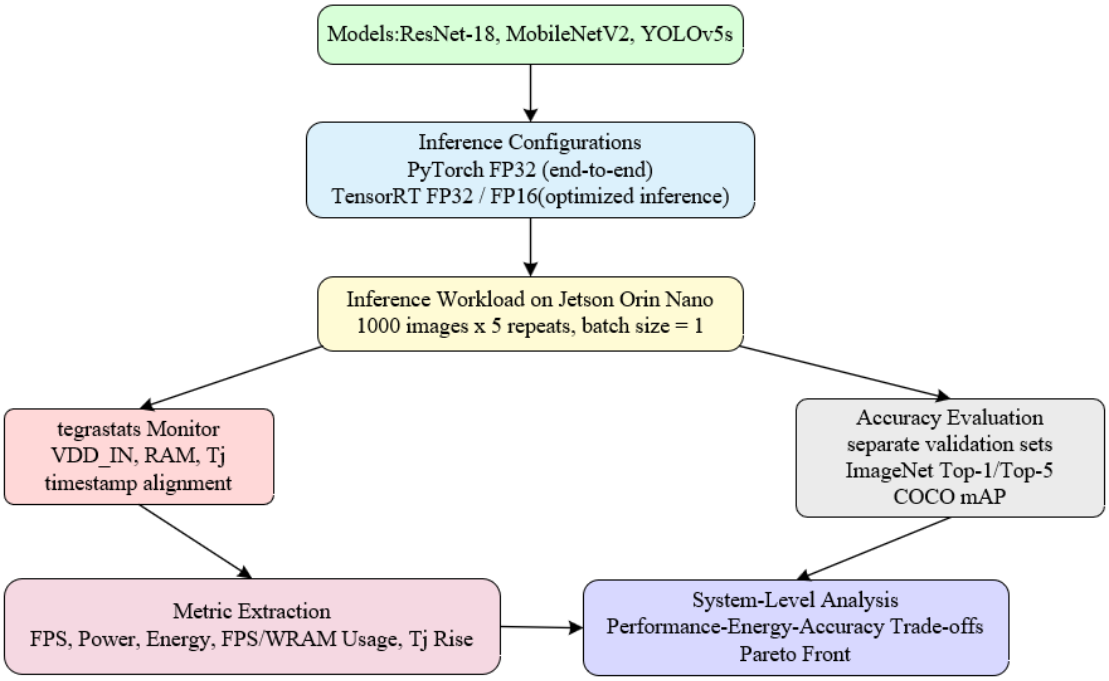

Figure 1: On Jetson Orin Nano, system-level DL inference evaluation uses tegrastats to log power, RAM, and temperature with monotonic timestamps. YOLOv5s TensorRT FP32/FP16 results are engine-level, not end-to-end pipeline measurements.

4. Results and Discussion

4.1. Inference Performance

Table 2 summarizes time and throughput for the 1000 x 5 workload. TensorRT greatly improves classification throughput: MobileNetV2 increases from 54.41 FPS to 337.58 FPS under TensorRT FP32, and ResNet-18 increases from 106.51 FPS to 373.48 FPS. YOLOv5s TensorRT values are engine-level results.

Table 2: Inference Performance Results.

Model	Configuration	Scope	Time (s)	FPS
ResNet-18	PyTorch FP32	End-to-end	46.943	106.512
	TensorRT FP32	End-to-end	13.388	373.475
	TensorRT FP16	End-to-end	14.041	356.104

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

Model	Configuration	Scope	Time (s)	FPS
MobileNetV2	PyTorch FP32	End-to-end	91.891	54.412
	TensorRT FP32	End-to-end	14.811	337.581
	TensorRT FP16	End-to-end	15.378	325.134
YOLOv5s	PyTorch FP32	End-to-end	101.907	49.064
	TensorRT FP32*	Engine-level	55.724	89.728
	TensorRT FP16*	Engine-level	30.387	164.546

*YOLOv5s TensorRT FP32/FP16 values are engine-level measurements and exclude parts of the Python loading, preprocessing, decoding, and NMS pipeline.

4.2. Explaining the FP16 Throughput Regression

TensorRT FP16 does not always improve throughput over TensorRT FP32. ResNet-18 decreases slightly from 373.48 FPS to 356.10 FPS, and MobileNetV2 decreases from 337.58 FPS to 325.13 FPS. Thus, FP16 is not automatically the fastest mode for small batch-size-1 workloads.

Telemetry suggests this is a system-level trade-off rather than an error. FP16 reduces average power and GPU activity for both classification models, moving execution toward a lower-power regime.

This behavior may result from kernel-launch overhead, memory movement, small model size, and power management. With batch size 1, these overheads can limit the arithmetic benefit of FP16. Some layers may also remain memory-bound.

Prior edge-device quantization profiling also shows that reduced precision can have model- and framework-dependent effects, so FP16 or INT8 modes should be validated empirically rather than assumed to dominate all metrics [10].

This observation is important because reduced precision is often assumed to be universally faster. The results show that the practical benefit of FP16 depends on the interaction between model size, batch size, memory traffic, runtime overhead, and device power management. For small batch-size-1 classification models, FP16 mainly shifts execution toward a lower-power regime, so FPS/W improves even when raw throughput does not increase.

Therefore, TensorRT FP16 should mainly be interpreted as an energy-efficiency mode in this study. FPS/W improves from 26.68 to 43.91 for ResNet-18 and from 34.41 to 44.54 for MobileNetV2.

4.3. Power Consumption and Energy Efficiency

Table 3 reports average power, GPU activity, energy, FPS, and FPS/W. TensorRT reduces total energy mainly by shortening the execution interval. MobileNetV2 energy decreases from 616.9 J to 145.3 J under TensorRT FP32 and 112.3 J under TensorRT FP16. Figure 2 summarizes throughput, energy, and FPS/W.

Table 3: Power, GPU Activity, Energy, and Efficiency Results.

Model	Configuration	Avg Power (W)	Avg GR3D (%)	Energy (J)	FPS	FPS/W
ResNet-18	PyTorch FP32	11.884	61.81	557.9	106.512	8.963
ResNet-18	TensorRT FP32	13.999	73.96	187.4	373.475	26.678
ResNet-18	TensorRT FP16	8.111	44.10	113.9	356.104	43.905
MobileNetV2	PyTorch FP32	6.713	43.66	616.9	54.412	8.105
MobileNetV2	TensorRT FP32	9.810	61.67	145.3	337.581	34.413
MobileNetV2	TensorRT FP16	7.300	50.23	112.3	325.134	44.536
YOLOv5s	PyTorch FP32	16.409	89.01	1672.2	49.064	2.990
YOLOv5s	TensorRT FP32*	17.484	93.67	974.3	89.728	5.132
YOLOv5s	TensorRT FP16*	16.470	93.55	500.5	164.546	9.991

*YOLOv5s TensorRT values are engine-level measurements and are not strict end-to-end comparisons with the PyTorch detection pipeline.

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

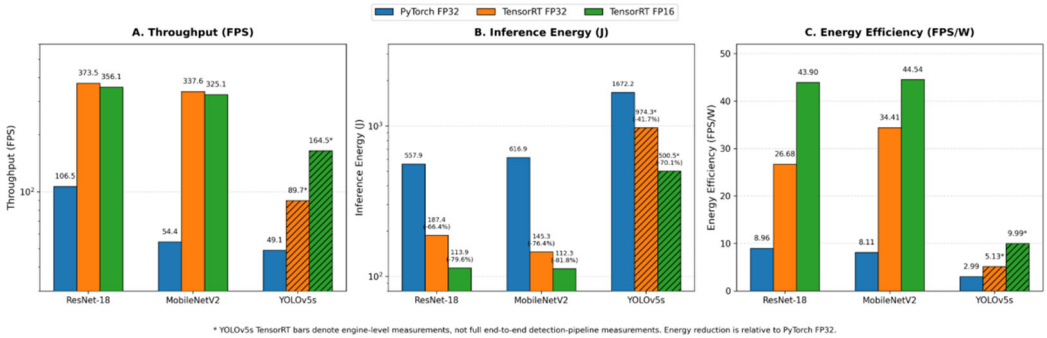

Figure 2: Key system-level metrics. Panel A shows FPS, Panel B shows energy for the 1000 x 5 workload, and Panel C shows FPS/W. Percentages in Panel B show energy reduction relative to PyTorch FP32. Hatched bars and asterisks mark YOLOv5s TensorRT engine-level results.

4.4. Power-Time Behavior

Figures 3-5 show that PyTorch FP32 maintains high power for longer periods, while TensorRT shortens the high-load interval. Because tegrastats samples every 500 ms, the traces show sustained workload-level behavior but may miss short power spikes.

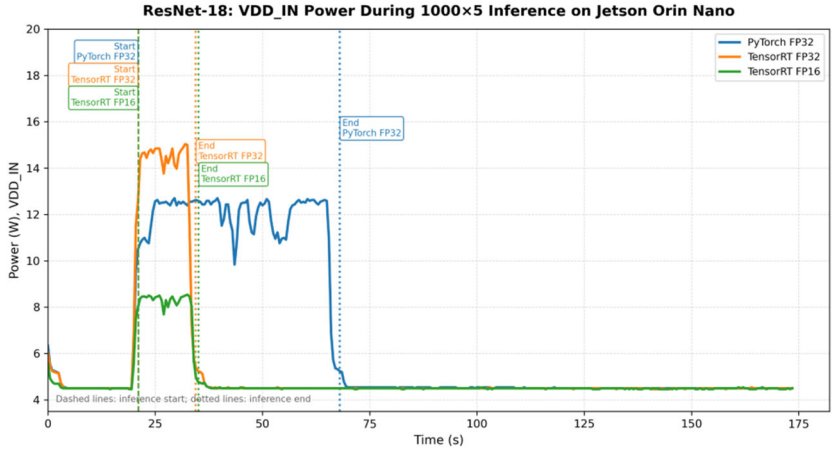

Figure 3: VDD_IN power trace for ResNet-18 during the 1000 x 5 workload. Dashed lines mark inference start times and dotted lines mark inference end times.

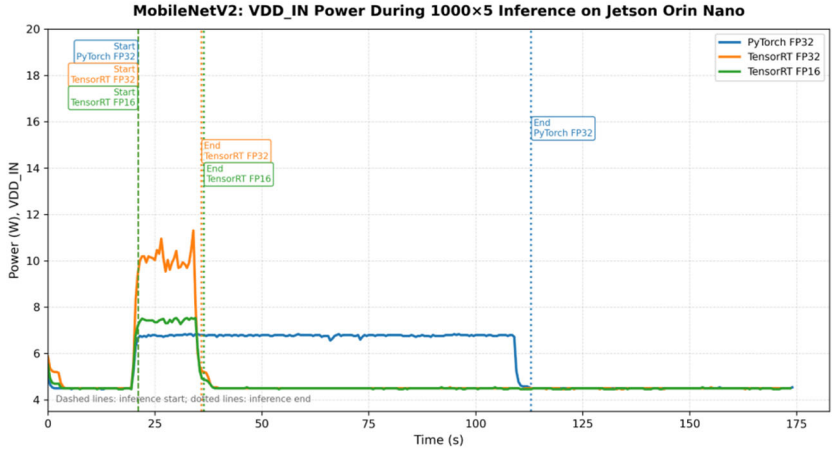

Figure 4: VDD_IN power trace for MobileNetV2 during the 1000 x 5 workload. TensorRT shortens the high-load interval compared with PyTorch FP32.

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

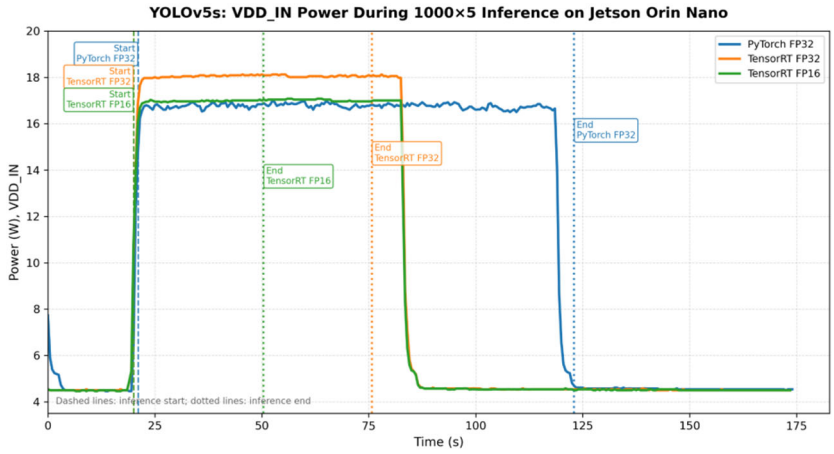

Figure 5: VDD_IN power trace for YOLOv5s during the 1000 x 5 workload. TensorRT FP16 has the shortest engine-level execution window.

4.5. Accuracy Evaluation

Table 4 shows accuracy. TensorRT FP32 preserves classification accuracy. ResNet-18 and YOLOv5s also retain accuracy under FP16, while MobileNetV2 Top-1 accuracy drops from 71.734% to 69.568%.

Table 4: Accuracy Evaluation Results.

Model	Task	Configuration	Top-1 (%)	Top-5 (%)	mAP@0.5	mAP@0.5:0.95
ResNet-18	Classification	PyTorch FP32	69.660	88.884	–	–
ResNet-18	Classification	TensorRT FP32	69.652	88.890	–	–
ResNet-18	Classification	TensorRT FP16	69.656	88.910	–	–
MobileNetV2	Classification	PyTorch FP32	71.734	90.110	–	–
MobileNetV2	Classification	TensorRT FP32	71.732	90.116	–	–
MobileNetV2	Classification	TensorRT FP16	69.568	88.966	–	–
YOLOv5s	Detection	PyTorch FP32	–	–	0.566	0.371
YOLOv5s	Detection	TensorRT FP32	–	–	0.565	0.371
YOLOv5s	Detection	TensorRT FP16	–	–	0.565	0.371

4.6. Energy–Accuracy Design Space

Table 5 lists the points used in the energy-accuracy analysis. Figure 6 shows the design space. TensorRT shifts points toward lower energy, but MobileNetV2 FP16 shows an accuracy cost. YOLOv5s TensorRT points are engine-level results.

The design-space view also clarifies the difference between harmless energy reduction and accuracy-costly optimization. ResNet-18 TensorRT FP16 is close to an ideal point because it greatly reduces energy while preserving Top-1 accuracy. MobileNetV2 TensorRT FP16, however, achieves the lowest energy but moves downward in accuracy, making it a clear example of a precision-dependent trade-off that should be validated before deployment.

Table 5: Energy–Accuracy Design-Space Points Used in the Pareto Analysis.

Model	Configuration	Energy (J)	Accuracy metric
ResNet-18	PyTorch FP32	557.9	69.660% Top-1
ResNet-18	TensorRT FP32	187.4	69.652% Top-1
ResNet-18	TensorRT FP16	113.9	69.656% Top-1
MobileNetV2	PyTorch FP32	616.9	71.734% Top-1

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

Model	Configuration	Energy (J)	Accuracy metric
MobileNetV2	TensorRT FP32	145.3	71.732% Top-1
MobileNetV2	TensorRT FP16	112.3	69.568% Top-1
YOLOv5s	PyTorch FP32	1672.2	37.1% mAP@0.5:0.95
YOLOv5s	TensorRT FP32*	974.3	37.1% mAP@0.5:0.95
YOLOv5s	TensorRT FP16*	500.5	37.1% mAP@0.5:0.95

*YOLOv5s TensorRT points are engine-level measurements and are not strict end-to-end comparisons with the PyTorch detection pipeline.

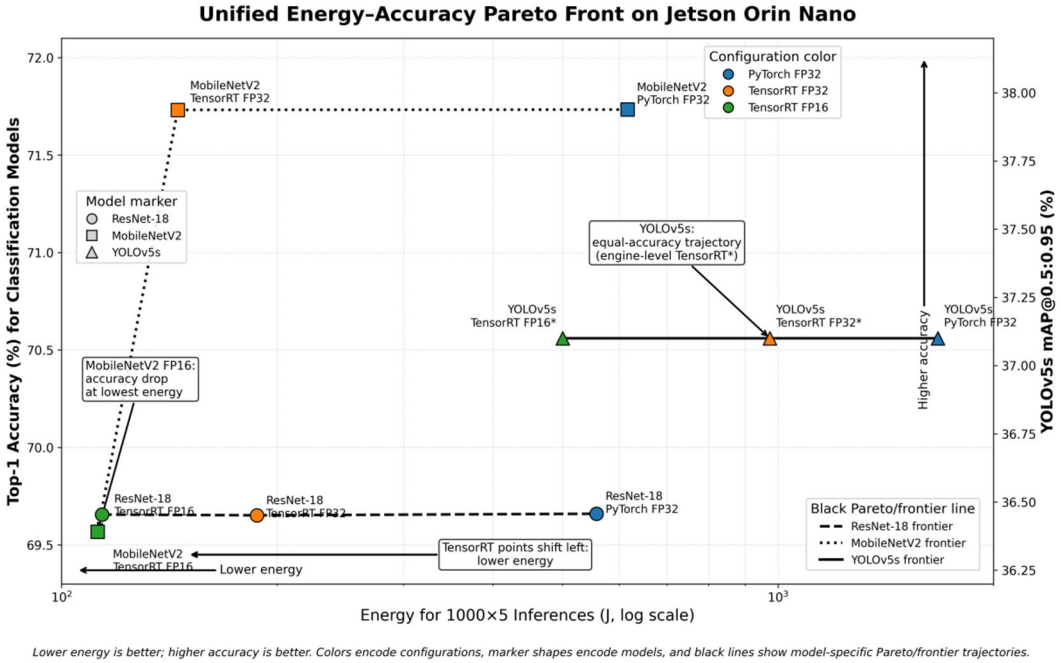

Figure 6: Energy-accuracy design space on Jetson Orin Nano. Colors indicate configurations, markers indicate models, and black lines show model-specific trajectories. Classification uses Top-1 accuracy; YOLOv5s uses mAP@0.5:0.95.

4.7. System RAM Usage

Table 6 shows RAM usage. TensorRT uses less system RAM than PyTorch FP32. YOLOv5s average RAM decreases from 4593.0 MB under PyTorch FP32 to 2020.3 MB under TensorRT FP16. No SWAP usage is observed.

Table 6: System RAM Usage during Inference.

Model	Configuration	Avg RAM (MB)	Peak RAM (MB)	Avg SWAP (MB)	Peak SWAP (MB)
ResNet-18	PyTorch FP32	4298.0	4314	0.0	0
ResNet-18	TensorRT FP32	3087.7	3103	0.0	0
ResNet-18	TensorRT FP16	3056.4	3072	0.0	0
MobileNetV2	PyTorch FP32	4483.7	4500	0.0	0
MobileNetV2	TensorRT FP32	3047.3	3063	0.0	0
MobileNetV2	TensorRT FP16	3039.4	3057	0.0	0
YOLOv5s	PyTorch FP32	4593.0	4594	0.0	0
YOLOv5s	TensorRT FP32	2060.0	2080	0.0	0
YOLOv5s	TensorRT FP16	2020.3	2040	0.0	0

4.8. Thermal Behavior

Table 7 and Figure 7 show temperature behavior. TensorRT reduces thermal accumulation for classification models; ResNet-18 Tj rise drops from 6.343 degrees C under PyTorch FP32 to 1.437

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

degrees C under TensorRT FP16. YOLOv5s remains hotter because its workload is heavier.

Temperature rise is included because it affects long-running edge applications such as cameras and robots. A configuration that finishes quickly can reduce thermal accumulation even if its instantaneous power is not always the lowest. The traces therefore complement the average-power table by showing how long the device remains in a high-load state during each workload.

This thermal view is especially relevant for fan-limited or enclosure-mounted edge devices, where repeated inference can gradually reduce available performance headroom even when the average FPS appears stable during a short benchmark.

Table 7: Tj Temperature Increase during Inference.

Model	PyTorch FP32	TensorRT FP32	TensorRT FP16
ResNet-18	6.343°C	4.344°C	1.437°C
MobileNetV2	2.094°C	2.125°C	0.875°C
YOLOv5s	14.812°C	12.625°C	11.532°C

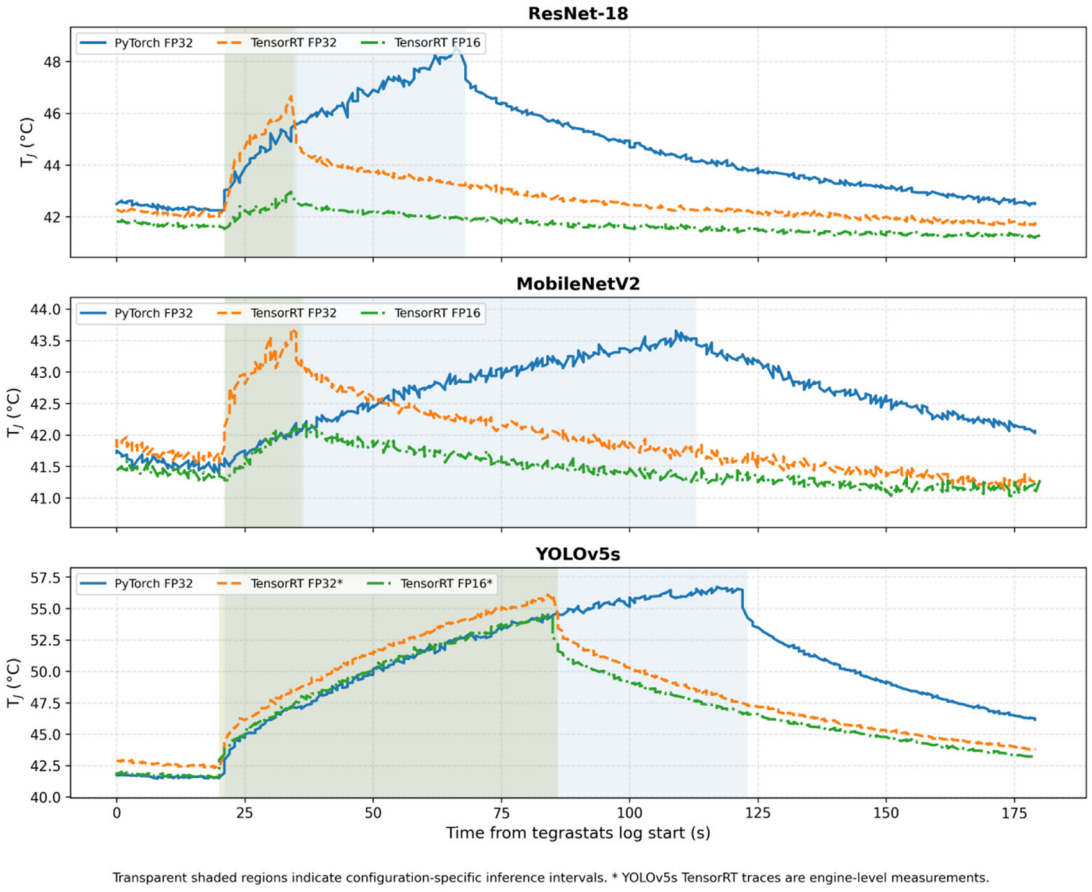

Figure 7: Tj temperature traces during the 180 s window. Shaded regions indicate inference intervals. TensorRT generally shortens high-load execution and reduces thermal accumulation for classification models.

5. Limitations and Deployment Implications

This study provides a system-level characterization of PyTorch and TensorRT inference on Jetson Orin Nano, but several limitations remain.

5.1. YOLOv5s TensorRT Measurement Scope

YOLOv5s TensorRT FP32/FP16 measurements are engine-level rather than fully matched end-to-end detection-pipeline measurements. The PyTorch baseline includes image loading, resizing, tensor preparation, network execution, decoding, and NMS. TensorRT results mainly measure engine

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

execution.

Thus, YOLOv5s TensorRT FPS and energy values should be interpreted as acceleration potential, not as strict end-to-end framework comparisons. A fully matched TensorRT detection pipeline is left for future work.

5.2. Power-Sampling Resolution

Power is sampled using tegrastats every 500 ms. This is sufficient for sustained workload comparison, but it may miss short spikes from kernel launches, memory transfers, or bursty GPU activity. Peak power and short transient energy may therefore be underestimated.

Energy is computed by aligning inference timestamps with tegrastats samples and applying trapezoidal integration. Higher-frequency external power measurement would be needed for sub-sample transient analysis.

5.3. Repeatability and Statistical Variation

The reported results are aggregate measurements over 5000 inferences. They describe workload-level averages rather than independent repeat-level variation. Future work will report standard deviation, range, and coefficient of variation.

5.4. GPU Frequency and DVFS Effects

Telemetry includes power, temperature, memory, and GPU activity, but detailed GPU frequency traces are not retained. Therefore, DVFS-related explanations are inferred from observed behavior rather than proven by direct clock traces.

5.5. Generality of the Results

The conclusions are specific to Jetson Orin Nano 8GB, the tested software stack, batch size 1, and the selected input resolutions. Other models, batch sizes, TensorRT versions, CUDA versions, and power modes may lead to different trade-offs.

6. Conclusion

This paper evaluated PyTorch FP32, TensorRT FP32, and TensorRT FP16 inference on Jetson Orin Nano using ResNet-18, MobileNetV2, and YOLOv5s. TensorRT substantially reduces energy and improves FPS/W. ResNet-18 FP16 reduces energy by 79.6% with nearly unchanged accuracy, while MobileNetV2 FP16 reduces energy by 81.8% but loses 2.166 percentage points of Top-1 accuracy. YOLOv5s TensorRT FP16 shows strong engine-level acceleration while maintaining mAP. Overall, edge AI optimization should be judged jointly by energy, accuracy, memory, temperature, measurement scope, and throughput.

References

[1] X. Wang et al., "A Comprehensive Survey on On-Device AI Models," ACM Computing Surveys, 2025.

[2] D. K. Alqahtani, A. Cheema, and A. N. Toosi, "Benchmarking Deep Learning Models for Object Detection on Edge Computing Devices," arXiv preprint, arXiv:2409.16808, 2024.

[3] S. P. Baller, A. Jindal, M. Chadha, and M. Gerndt, "DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices," arXiv preprint, arXiv:2108.09457, 2021.

[4] NVIDIA, "Jetson Orin Nano Super Developer Kit," NVIDIA Embedded Systems, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/. Accessed: Jul. 3, 2026.

[5] NVIDIA, "NVIDIA TensorRT Developer Guide," NVIDIA Developer Documentation, 2023. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/. Accessed: Jul. 3, 2026.

[6] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2019.

ISSN 2616-5775 Vol. 9, Issue 7: 9-19, DOI: 10.25236/AJCIS.2026.090702

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2016, pp. 770–778.
[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), 2018, pp. 4510–4520.
[9] G. Jocher et al., "YOLOv5 by Ultralytics," Zenodo, 2020, doi: 10.5281/zenodo.3908559.
[10] H. Ahn, T. Chen, N. Alnaasan, A. Shafi, M. Abduljabbar, H. Subramoni, and D. K. Panda, "Performance Characterization of Using Quantization for DNN Inference on Edge Devices," arXiv preprint, arXiv:2303.05016, 2023.