ADAPTIVE EDGE AI CONTROLLER

Thermal-Aware Adaptive Workload Control for Jetson-Class Edge AI Devices

ATATÜRK UNIVERSITY

Faculty of Engineering

Report Type	**Edge AI / Computer Vision / Thermal-Aware Control**
Primary Platform	NVIDIA Jetson-class edge AI devices
Primary Workload	YOLO-based real-time inference
Experimental Evidence	3 CSV datasets, 7 existing figures and additional analysis plots
Report Date	15 May 2026

15 May 2026

Technical Supervisor: Assoc. Prof. Dr. Barış Özyer

Department of Computer Engineering

Prepared by: Adaptive Edge AI Controller Research Team

İrfan Çötoğlu

Efsa Uzun

ETHICS AND COPYRIGHT STATEMENT

The work, analyses, figures and engineering interpretation presented in this report have been prepared by the project team. External sources, documentation and user reports used to establish the technical context are cited in the References section. The report has been prepared in accordance with academic integrity principles and is intended to document the design, implementation and experimental evaluation of the Adaptive Edge AI Controller system.

Author confirmation:

İrfan Çötoğlu:

Efsa Uzun:

Fatih Ayıbasan:

Contents

Abstract and Executive Summary

This report presents a thermal-aware adaptive workload control mechanism for a YOLO-based real-time human-detection pipeline running on Jetson-class edge AI devices. The central objective is to scale the inference workload in response to device temperature: when the thermal state deteriorates, the controller gradually reduces the inference image size and the percentage of processed frames; when the system returns to a safe thermal band, it restores performance in a controlled manner.

The motivation is not limited to raw inference speed. In security cameras, robotic platforms, unmanned aerial systems, outdoor sensor nodes and other unattended edge AI deployments, heat build-up can lead to lower FPS, increased latency, thermal throttling, system freezes, automatic shutdown and higher maintenance requirements. NVIDIA positions Jetson platforms for robotics, drones, video analytics and autonomous machines [W1]-[W2], while developer forums frequently report temperature rise, latency accumulation, freezing and shutdown under heavy YOLO, DeepStream and TensorFlow inference workloads [F1]-[F10].

3,900 Demo samples	**130 min** Demo duration	**52.6-81.8 °C** Temperature range	**10.0%** Time above 80 °C	**-1.85 °C** FPS experiment temperature change

In the closed-loop demonstration experiment, the system ran for approximately 130 minutes and recorded 3,900 telemetry samples. GPU temperature ranged from 52.56 °C to 81.82 °C. The 70 °C threshold was crossed at approximately minute 22.1, and the 80 °C threshold at approximately minute 72.3. The system did not exceed 85 °C, and the emergency mode was not triggered. This indicates that the SafetyGuard layer and the adaptive throttling decisions were effective in keeping the system below the hard critical band. At the same time, the fact that approximately 10.0% of the experiment was spent above 80 °C shows that earlier intervention strategies, fan telemetry and power telemetry could further improve stability.

In the FPS-scaling experiment, the processed-frame ratio was reduced stepwise from percentage = 1.0 to percentage = 0.25. This produced an average GPU temperature reduction of approximately 1.85 °C. In the final ten minutes, which are closer to thermal steady state, the temperature difference increased to 2.19 °C. In the same experiment, average GPU load fell from 72.9% to 26.4%, while average CPU load fell from 19.1% to 10.7%. These results show that reducing the proportion of processed frames lowers both the thermal load and processor utilisation in the inference pipeline.

DEMONSTRATION VIDEO: Adaptive Edge AI Controller Results Demonstration - Results Video

1. Problem Definition and Motivation

Edge AI systems combine cameras, sensors and GPU-accelerated inference hardware on the same physical platform so that decisions can be made locally without sending all data to a central server. This architecture reduces latency and dependency on network connectivity, but thermal stability becomes a central engineering problem because compact devices operate with limited cooling capacity, constrained power budgets and sustained computational loads.

Real-time object detectors such as YOLO continuously exercise GPU, CPU, memory and image-processing resources, especially under continuous video streaming, multi-camera operation or high-resolution inference. As temperature increases, operating-system and firmware-level mechanisms such as fan management, dynamic voltage and frequency scaling (DVFS) and clock throttling are activated. NVIDIA Jetson Linux documentation states that the BSP uses fan control and clock throttling for thermal cooling, and that reducing clock frequency can directly affect performance and user experience [W5].

1.1 Link to Field-Reported User Problems

The problem is not merely theoretical. Developer forums and open-source issue reports show that overheating, shutdown, freezing, latency accumulation and FPS loss recur in practical Jetson and edge AI deployments. The table below relates observations from the reviewed sources to the engineering problem addressed by this work.

Source	User observation	Relevance to this work
[F1]	On a Jetson Nano running YOLOv8, temperature rises to approximately 70 °C after around ten minutes, creating risk for outdoor autonomous use.	Adaptive control should account for temperature trend in long-duration outdoor operation.
[F2]	During real-time object detection from an RTSP stream, the device shuts down after 15-20 minutes and the heatsink becomes excessively hot.	Model execution alone is insufficient; operational continuity and thermal safety must be handled together.
[F3]	Under multi-camera YOLO inferencing, the system freezes completely within one to two hours and requires manual restart.	Freezing that requires manual intervention is a critical operational risk for unattended or remote systems.
[F4]	A DeepStream pipeline shows temperature and latency accumulation; drop-frame/interval settings reduce both temperature and latency.	The FPS or processed-frame ratio is a practical thermal-control lever that directly changes system behaviour.
[F7]	High-performance mode causes crashes; 5 W mode reduces crashes while also reducing FPS.	There is an unavoidable trade-off among power, performance and stability; the software layer should manage this trade-off explicitly.
[F8]	TensorFlow object detection produces over-current throttling messages alongside temperature increase.	Thermal control should be considered together with power budget; reducing workload can lower both temperature and current/power risk.

1.2 Operational Criticality

- In a security-camera scenario, lower FPS or higher latency can delay event detection; a system freeze can interrupt the video stream entirely.
- In robotics, a slower perception pipeline reduces the freshness of environmental information used for awareness and safe motion planning.
- On drones and outdoor platforms, active cooling may not always be sufficient because of weight, energy, vibration and ambient-temperature constraints.
- In remote or long-running installations, shutdowns and freezes can create maintenance and restart costs that require physical access.

- When thermal throttling is triggered only at the system level, the application has limited control over which quality dimension degrades and by how much; this work makes that degradation controlled and measurable.

2. Jetson and Edge AI Deployment Context

NVIDIA Jetson is a widely used family of low-power, GPU-accelerated edge AI platforms. NVIDIA positions the Jetson Orin Nano Developer Kit for entry-level AI robots, intelligent drones and smart cameras [W1]. NVIDIA also emphasises the use of the Jetson platform in robotics, drones, intelligent video analytics and autonomous machines [W2]. For Jetson Nano, NVIDIA highlights support for neural-network workloads such as object detection, image classification, segmentation and speech processing within a low-power envelope [W3].

Within this context, the Adaptive Edge AI Controller is designed to preserve the local inference capability that makes Jetson-class devices valuable while making that capability more sustainable during extended operation. A highly accurate edge AI application loses field value if the device slows down, freezes or shuts down for thermal reasons. Thermal awareness is therefore not only a hardware-cooling issue; it is also an application-layer system-design requirement.

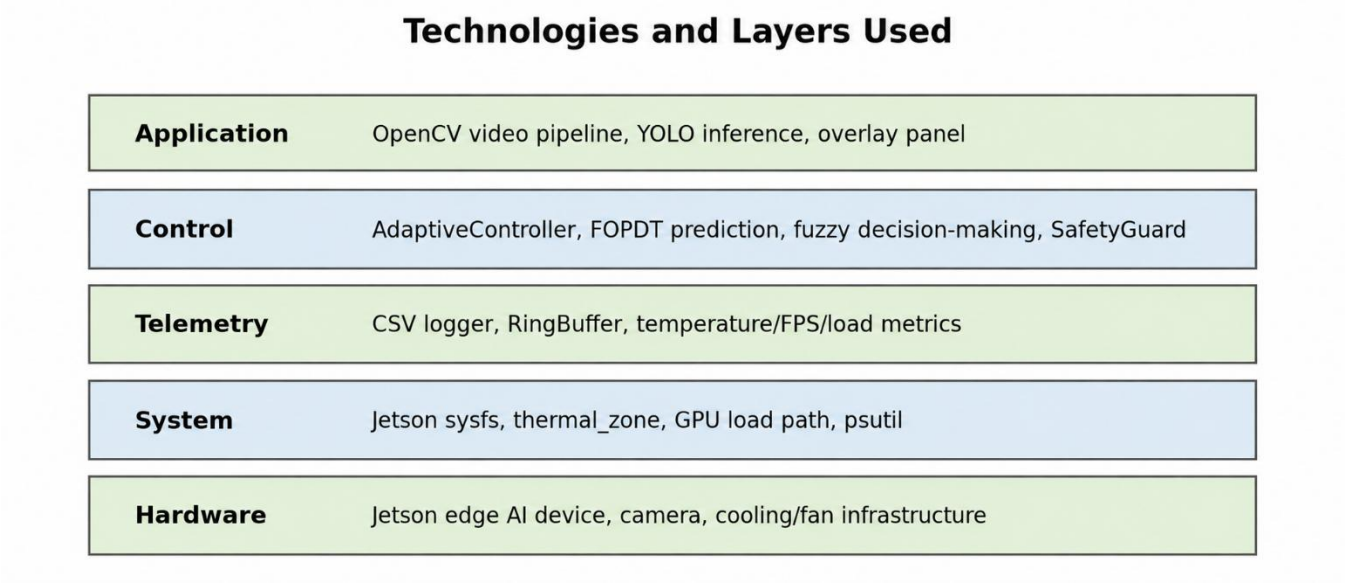

Figure 1. Core technology layers used in the system.

3. Objective, Scope and Contributions

3.1 Objective

The objective of the system is to automatically adapt a YOLO-based real-time human-detection pipeline running on a Jetson-class edge AI device according to thermal state. The controller monitors GPU temperature and temperature trend, estimates near-future thermal risk using an FOPDT-based predictor, selects the inference resolution and processed-frame ratio through a fuzzy-logic decision layer, and applies fail-safe actions through SafetyGuard when critical thresholds are reached.

3.2 Scope

- Real-time image acquisition from a camera stream and human detection using a YOLO model.
- CSV logging of telemetry fields including GPU temperature, GPU load, CPU load, FPS, imgsz, percentage and control mode.
- Execution of the closed-loop controller as a background thread.

- FOPDT thermal prediction, fuzzy-logic decision-making and a critical-threshold safety layer.
- Measurement of the thermal effect of the FPS/percentage and resolution/imgsz control levers through controlled experiments.

3.3 Contributions

Contribution	Description
Application-level thermal throttling	The application reduces its own workload before kernel-level automatic clock throttling becomes dominant.
Controlled quality degradation	When temperature rises, the system lowers resolution and processed-frame ratio through defined operating levels rather than degrading unpredictably.
Proactive decision-making	The controller uses not only instantaneous temperature but also temperature trend and predicted future temperature.
Experimental validation	Behaviour is quantified through CSV data from the demonstration, FPS and resolution experiments.
Modular codebase	Sensor, control, telemetry and demonstration layers are separated, supporting sustainable development and future extension.

4. Technologies and Implementation Stack

Technology / Component	Use in the System	File / Evidence
Python	Main application, control loop, sensor reading and experiment scripts.	thermal_edge/*, examples/basic_yolo_jetson.py
Ultralytics YOLO	Model inference for human detection in camera frames.	examples/basic_yolo_jetson.py, insan_tespit_mdeli.pt
PyTorch / CUDA	Execution of the YOLO model on the Jetson GPU.	torch.cuda.is_available() inside load_model()
OpenCV + GStreamer	Jetson CSI camera pipeline and real-time visualisation.	gstreamer_pipeline(), cv2.VideoCapture
Jetson sysfs	File-system based reading of GPU temperature and GPU load.	/sys/class/thermal, /sys/devices/gpu.0/load
psutil	CPU utilisation measurement.	read_cpu_load(), experiment logger classes
NumPy	Linear fit for the temperature trend over the last 30 seconds.	controller.py _compute_temp_delta()
scikit-fuzzy	Construction of the fuzzy-logic decision system.	control/fuzzy.py
CSV telemetry	Repeatable storage and analysis of experiment data.	adaptive_edge_demo.csv, fps_deney.csv, resolution_deney.csv
YAML configuration	Target temperature, critical thresholds, camera, YOLO and logging settings.	config.py, examples/configs/default.yaml

NVIDIA provides the tegrastats utility for reporting memory and processor statistics on Jetson devices [W4]. This implementation reads temperature and GPU-load signals directly from sysfs paths instead of parsing tegrastats output. That choice keeps the runtime lightweight and dependency-minimal. However, as discussed in the improvement section, the control loop should be enriched with power/current, throttling-state and fan-speed signals through tegrastats or the corresponding system interfaces in a future deployment-grade version.

5. System Architecture and Code Review

The implementation is structured as a modular Python package. Sensor acquisition, telemetry logging, decision logic and the demonstration application are separated into distinct components. This separation improves testability, maintainability and portability across Jetson-class devices.

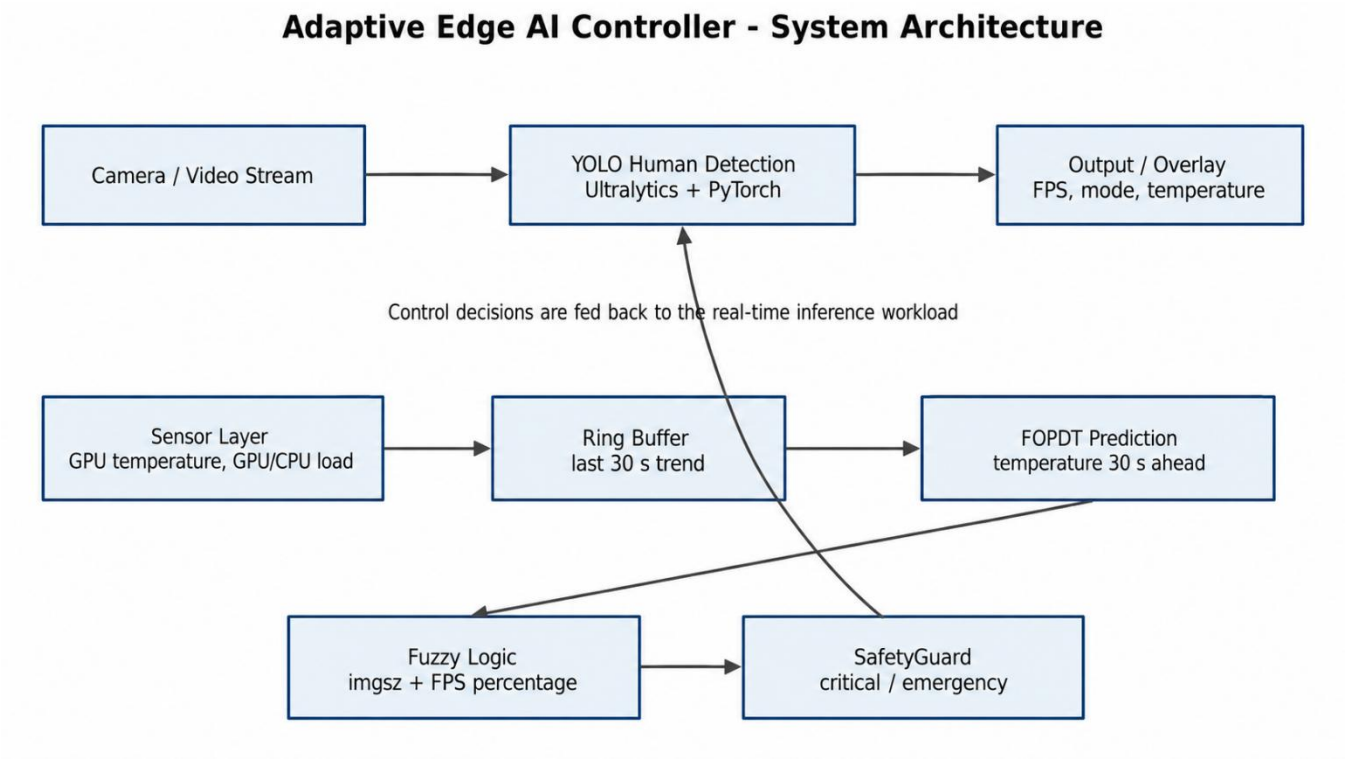

Figure 2. Adaptive Edge system architecture: video pipeline, telemetry, prediction, decision-making and feedback.

5.1 Package Structure

Layer	Primary Files	Responsibility
Sensor	thermal_zone.py, gpu_load.py	Safely read GPU temperature, GPU load and CPU load.
Telemetry	logger.py, ring_buffer.py	CSV logging and in-memory retention of recent samples.
Control	controller.py, fopdt.py, fuzzy.py, safety.py	Generate thermal decisions, predict temperature and enforce critical-threshold protection.
Demo	examples/basic_yolo_jetson.py	Camera stream, YOLO inference, frame skipping, overlay and control integration.
Experiments	fps_deney.py, imgsz(resolution)_deney.py	Measure the thermal effect of control levers through single-variable experiments.
Tests	tests/manual/*.py	Mock control loop, FOPDT checks and demonstration preflight controls.

5.2 Code Quality Assessment

Criterion	Observation	Assessment
Modularity	Sensor, control, telemetry and application layers are separated into distinct files.	✔
Thread safety	AdaptiveController protects state fields with a Lock.	✔
Error handling	Fallback returns are provided when sensor files are absent; import errors include explanatory messages.	✔
Configuration	Dataclass-based configuration with YAML section mapping.	✔
Testability	Mock loop and FOPDT tests exist; automated CI/pytest coverage can be extended.	✔
Portability	Jetson sysfs paths include discovery/fallback logic; sensor mapping across models can be extended.	✔

6. Control Mechanism

The control mechanism has three layers: FOPDT-based thermal prediction, fuzzy-logic action selection and SafetyGuard-based safety validation. The control loop runs every two seconds by default. At each iteration, it reads temperature, GPU load, CPU load and the latest FPS value; updates the ring buffer; computes trend and prediction; generates a control decision; and writes telemetry to CSV.

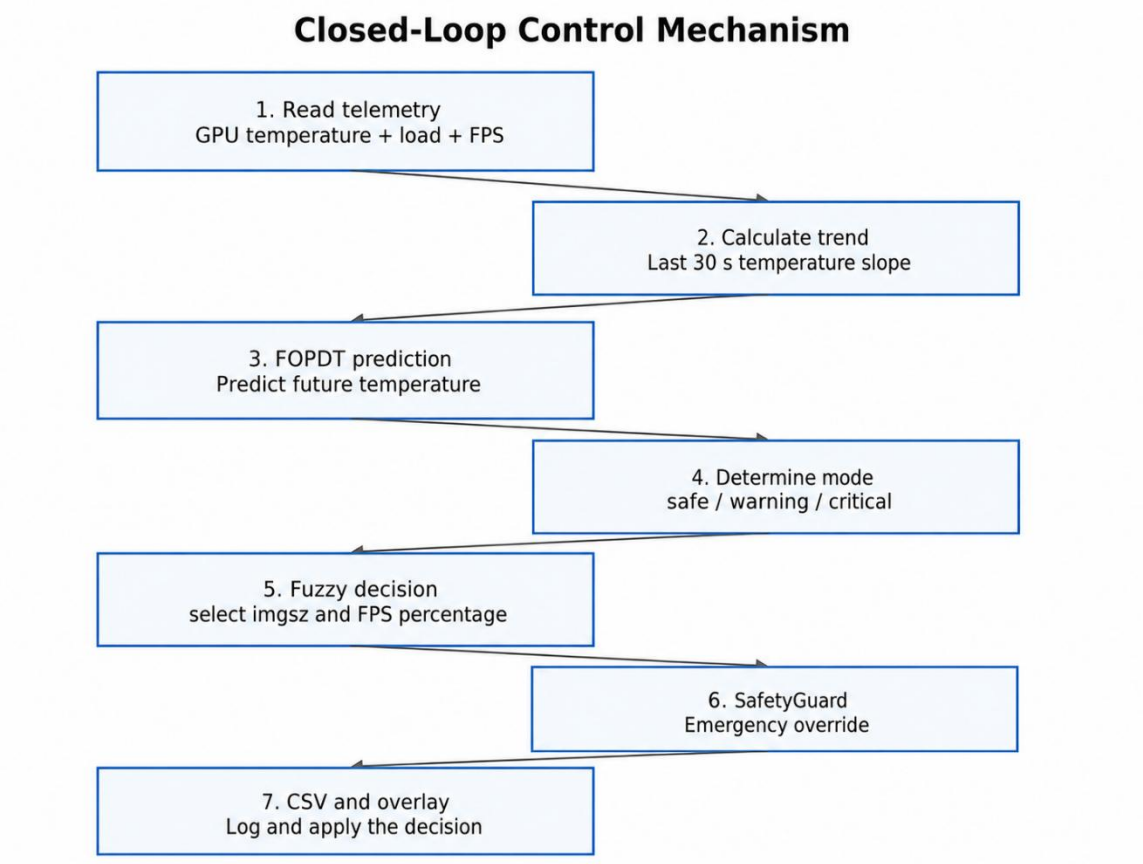

Figure 3. Execution flow of the control loop.

6.1 FOPDT Thermal Prediction

The FOPDT (First Order Plus Dead Time) approach approximates thermal dynamics using a time constant and a dead-time term. In the implementation, the prediction horizon is configured as 30 seconds, dead time as 2 seconds, the time constant as 60 seconds and the maximum prediction delta as 10 °C. The underlying model can be summarised as follows:

$$G(s) = \frac{K \cdot e^{-Ls}}{\tau s + 1}$$

FopdtThermalPredictor uses the temperature trend to estimate a future temperature delta and selects the control temperature as max(current_temp, predicted_temp). This conservative formulation helps the controller initiate throttling earlier when a rising temperature trend indicates near-future risk.

6.2 Fuzzy-Logic Decision Layer

ThermalFuzzyController uses two inputs: the deviation from the target temperature (temp_error) and the temperature trend (temp_delta). It produces imgsz_level and fps_level outputs, which are then mapped to discrete operating points: 320/480/640 px resolution and 25%/50%/75%/100% processed-frame percentage. A 10-second minimum action interval reduces oscillation and prevents the controller from changing decisions too frequently.

Temperature state	Trend	Typical decision
Safe	Falling / stable / rising	640 px, 100% - maximum performance
Warning	Falling or stable	640 px, 75% - light FPS restriction
Warning	Rising	480 px, 50% - moderate restriction
Critical	Falling or stable	320 px, 50% - aggressive resolution reduction
Critical	Rising	320 px, 25% - maximum software-level restriction

6.3 SafetyGuard and Fail-Safe Behaviour

SafetyGuard acts as the final safety filter on every control action. The default thresholds are 65 °C target temperature, 80 °C critical temperature and 85 °C hard-critical temperature for emergency behaviour. Above 85 °C, the emergency action is set to 320 px and 25%. The system leaves emergency mode only after the temperature drops at least 5 °C below the hard-critical threshold and remains in that region for 30 seconds. This hysteresis prevents rapid mode switching around the threshold.

7. Experimental Design and Data Collection

The project archive contains three primary experimental datasets. The experiments focus on collecting temperature and performance telemetry during YOLO-based human detection on Jetson-class hardware. CSV fields were selected to relate temperature, GPU/CPU load and the active performance parameter in each experiment.

Experiment	CSV File	Samples	Duration	Primary purpose
Demo / Closed loop	adaptive_edge_demo.csv	3900	130.0 min	Observe mode/action behaviour during long-running closed-loop control.
FPS / percentage	fps_deney.csv	7087	60.0 min	Measure temperature and load change when the processed-frame ratio is reduced.
Resolution / imgsz	resolution_deney.csv	7127	60.0 min	Measure temperature, FPS and load change when input resolution is reduced.

The timestamp values in the FPS and resolution experiments appear to start from 1970, which may indicate that the device clock was not synchronised. For this reason, the analyses use elapsed time relative to the first sample rather than absolute date/time. The demonstration CSV contains real date-time stamps.

8. Experimental Results and CSV Analysis

8.1 Demonstration Experiment: Closed-Loop Adaptive Control

In the demonstration experiment, the system ran for approximately 130 minutes and produced 3,900 telemetry rows at an average sampling interval of 2 seconds. Temperature started at 52.56 °C and reached a maximum of 81.82 °C. The 70 °C threshold was crossed at approximately minute 22.1, and the 80 °C threshold at approximately minute 72.3. The system did not exceed 85 °C.

Metric	Value
Number of records	3900
Duration	130.0 min
Sampling interval	2.00 s
GPU temperature min/mean/max	52.56 / 74.17 / 81.81 °C
Average FPS	17.09
Average GPU/CPU load	53.3% / 14.8%
Time ratio above 70 °C	81.4%
Time ratio above 80 °C	10.0%

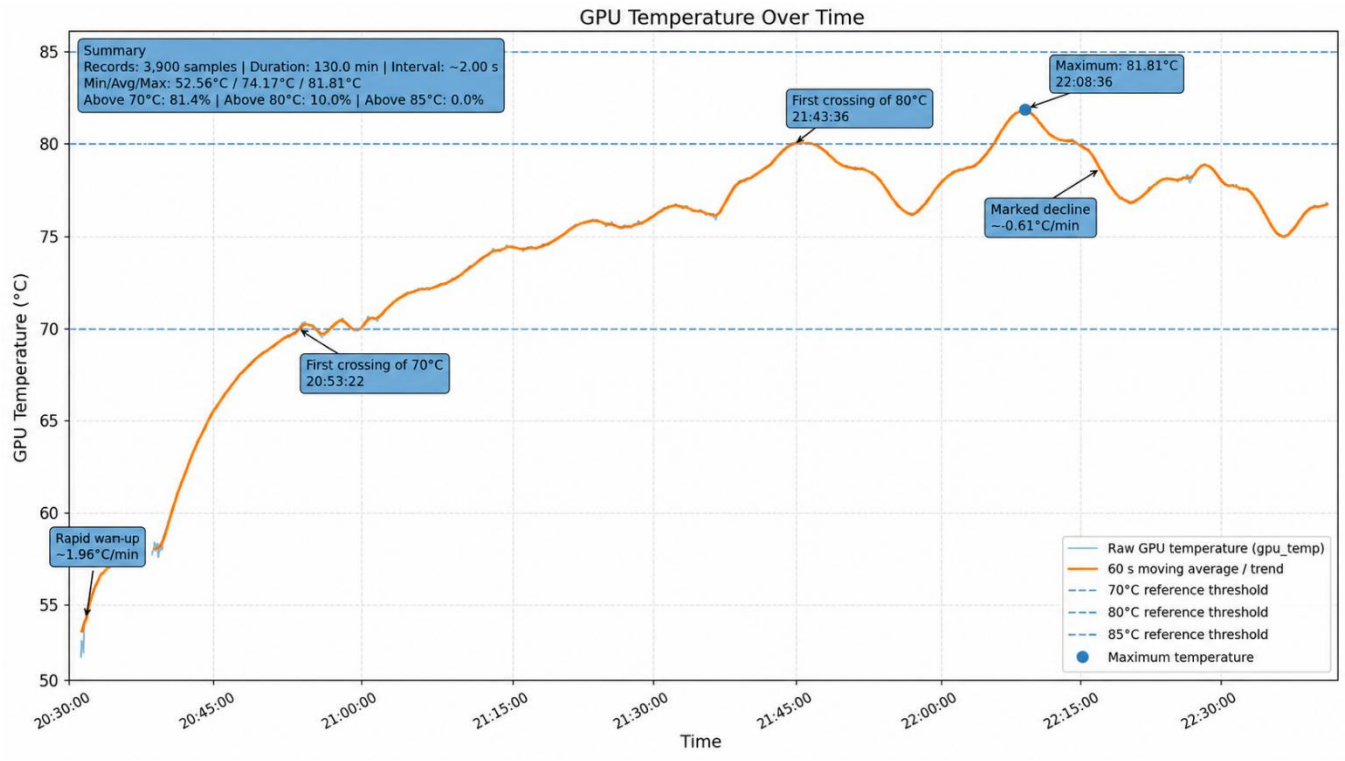

Figure 4. Demonstration figure from the project archive: GPU temperature over time and threshold crossings.

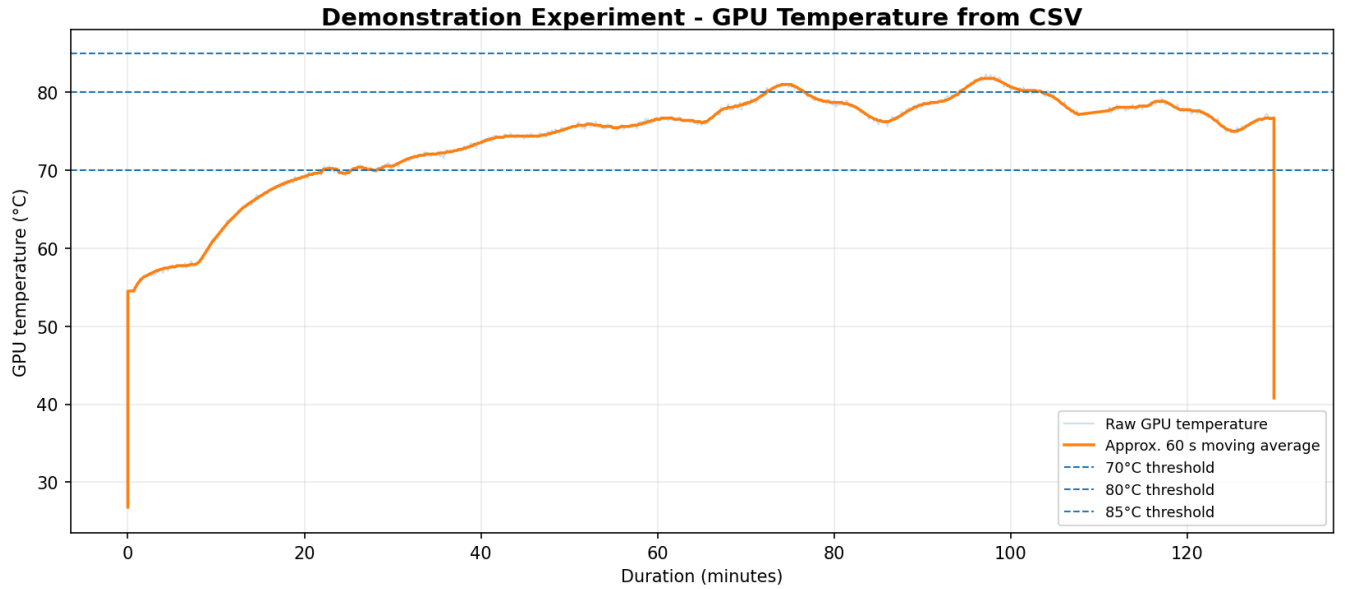

Figure 5. Demonstration temperature time series reproduced from the CSV data.

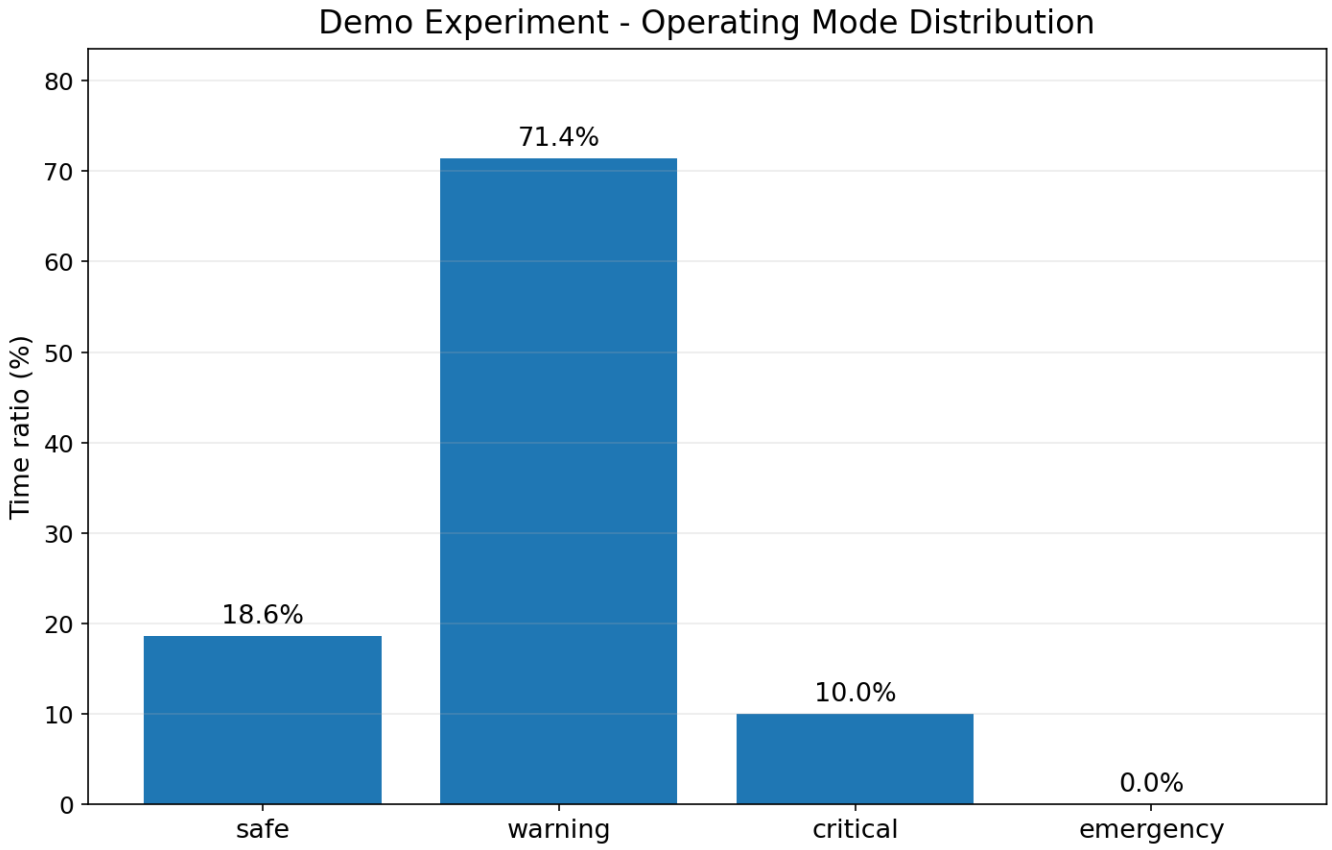

Figure 6. Safe/warning/critical mode distribution during the demonstration experiment.

Mode	Rows	Ratio	Mean temperature	Mean FPS	Mean imgsz	Mean percentage	Mean GPU load
critical	390	10.0%	80.79 °C	11.07	320	0.35	32.1%
safe	726	18.6%	63.35 °C	23.61	640	1.00	73.5%
warning	2784	71.4%	76.06 °C	16.23	539	0.59	51.0%

Mode-level results clearly show the controller behaviour: average FPS was 23.61 in safe mode, 16.23 in warning mode and 11.07 in critical mode. In safe mode, imgsz and percentage represent maximum performance; in critical mode, average imgsz falls to 320 and average percentage to approximately 0.35. This demonstrates that the controller reduces inference workload as temperature increases.

Action	Usage count	Interpretation
480 px / 50%	1750	Moderate restriction
640 px / 75%	1029	Moderate restriction
640 px / 100%	727	Maximum performance
320 px / 25%	235	Aggressive restriction
320 px / 50%	159	Aggressive restriction

The demonstration data show a negative correlation between temperature and FPS. This should not be interpreted as the physical claim that lowering load increases temperature. Instead, it reflects closed-loop feedback behaviour: as temperature rises, the controller lowers the FPS percentage and resolution, so lower FPS is observed in high-temperature regions. The correlation is therefore not a causal thermal law but a signature of the feedback policy.

8.2 FPS / Percentage Experiment

The purpose of this experiment was to quantify the effect of FPS percentage on temperature and to provide evidence for the FOPDT model parameters associated with the percentage control lever. The resolution was held constant at 640 px, while percentage was reduced from 1.0 to 0.25 halfway through the experiment. This isolates the effect of reducing the processed-frame ratio on temperature and system load.

Phase	n	Duration	Mean FPS	Mean GPU load	Mean CPU load	Mean temperature	Min-Max temperature
100%	3526	29.7 min	24.06	72.9%	19.1%	56.47 °C	51.03-57.75 °C
25%	3561	30.0 min	8.34	26.4%	10.7%	54.63 °C	53.84-57.34 °C

Comparison	Mean difference	Last 10 min difference
GPU temperature	-1.85 °C	-2.19 °C
FPS	-15.72	-15.71
GPU load	-46.5 points	-47.1 points
CPU load	-8.4 points	-8.4 points

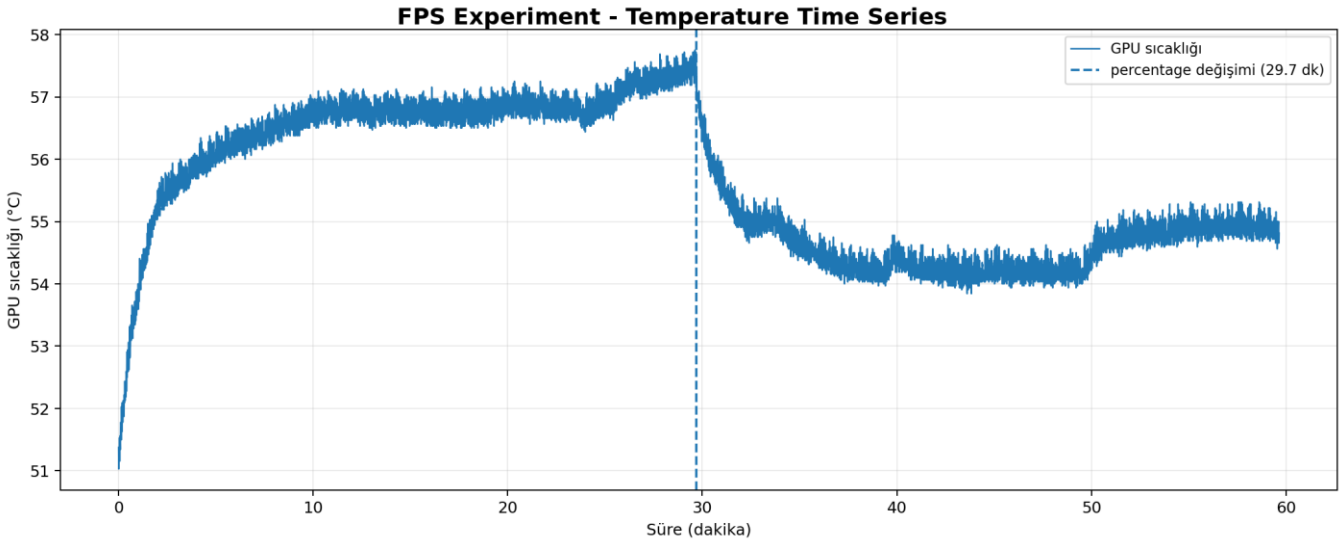

Figure 7. FPS experiment: temperature time series after the percentage change.

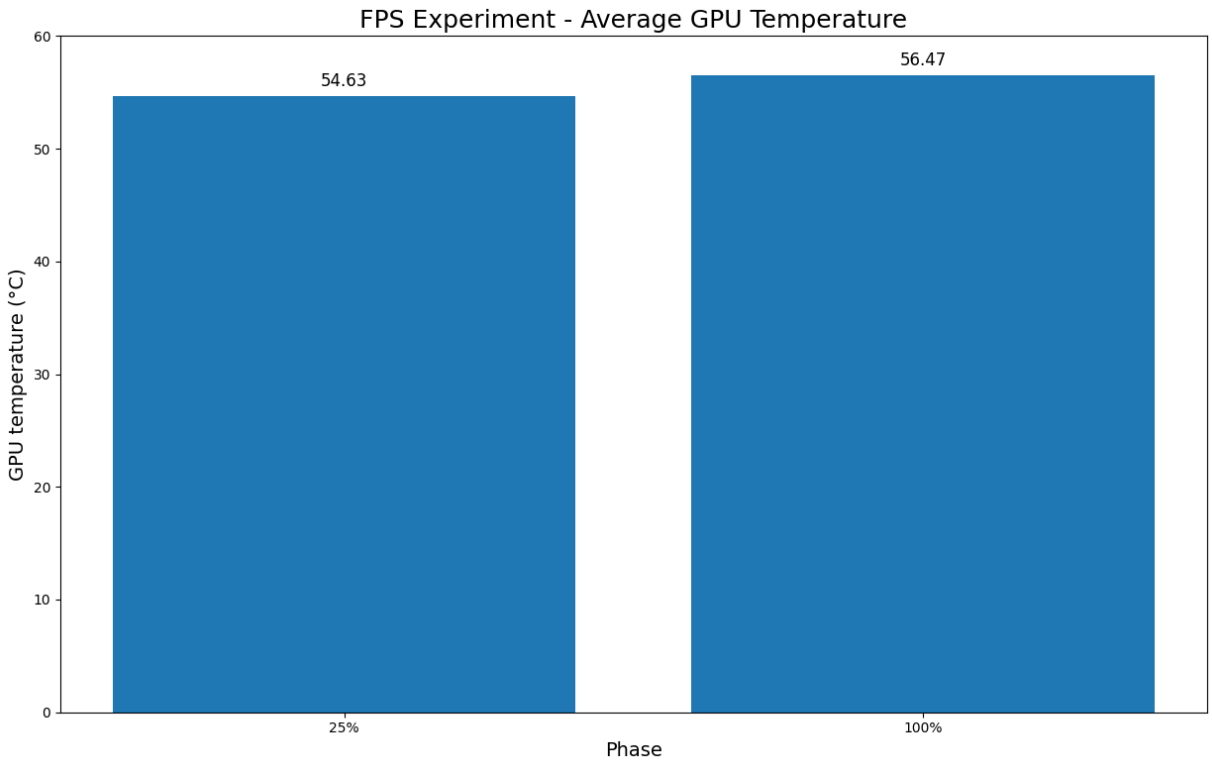

Figure 8. FPS experiment: average GPU temperature by percentage phase.

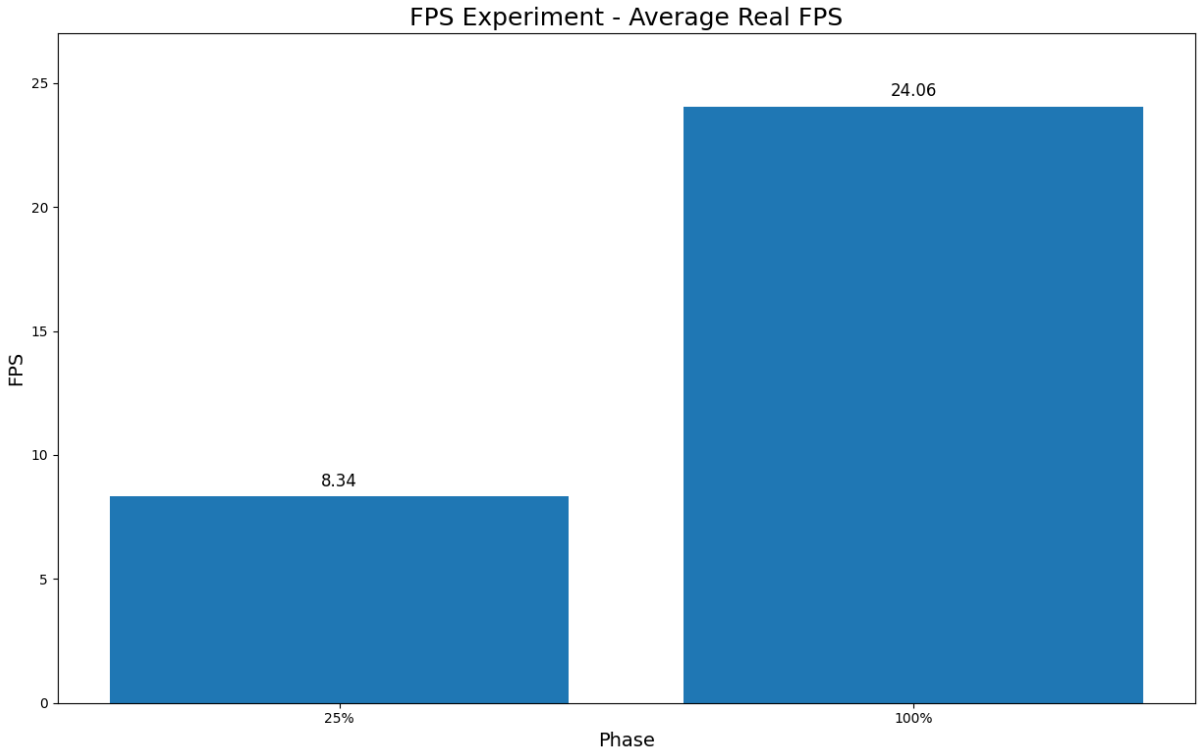

Figure 9. FPS experiment: average realised FPS by percentage phase.

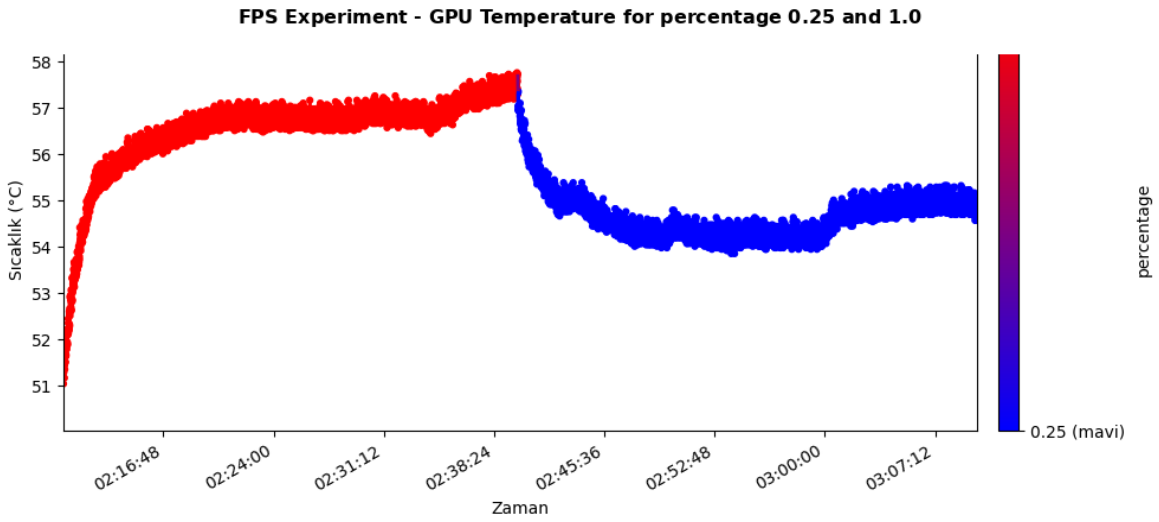

Figure 10. FPS experiment temperature figure from the project archive.

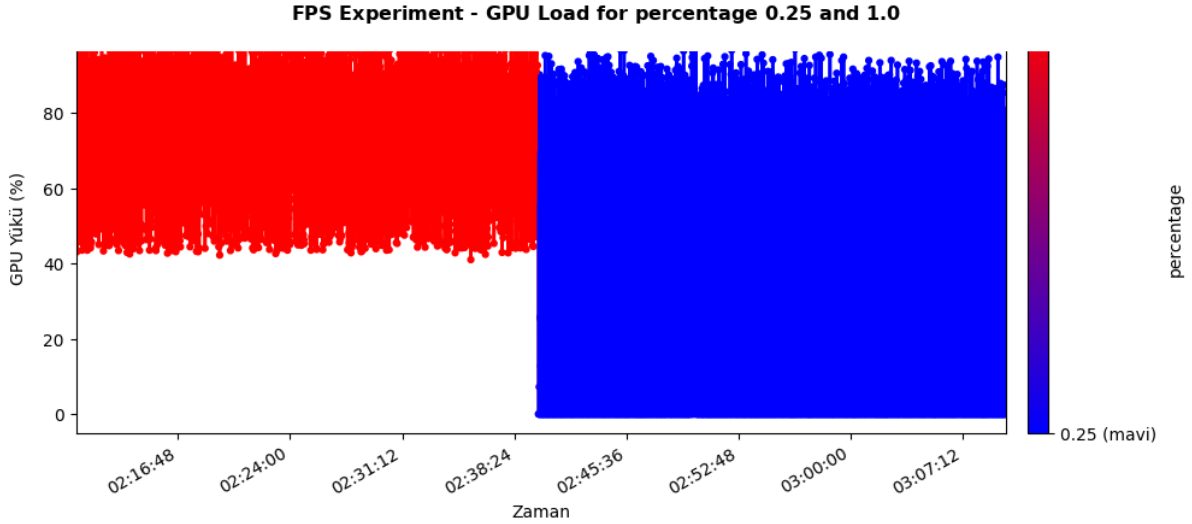

Figure 11. FPS experiment GPU-load figure from the project archive.

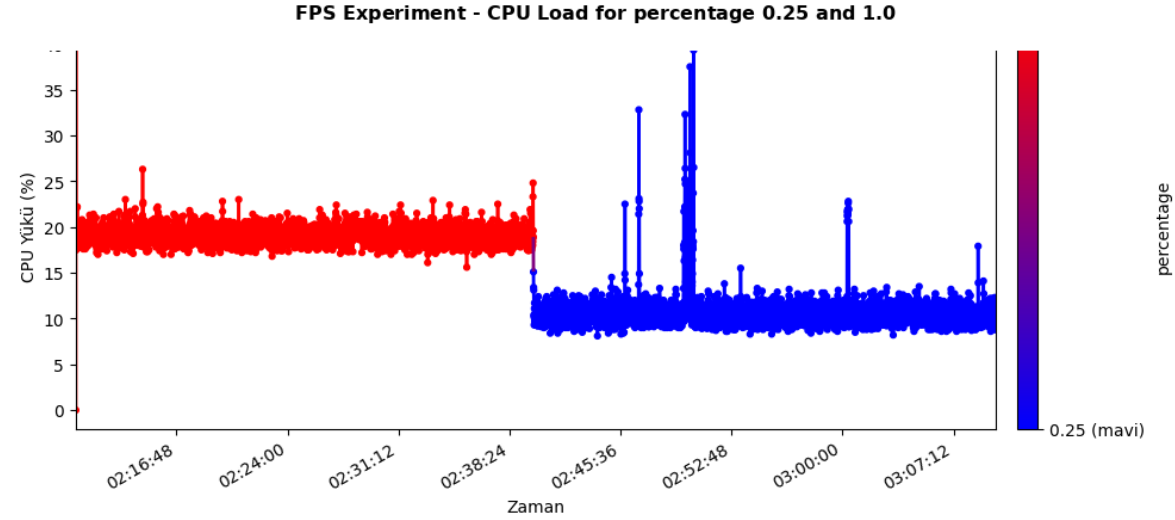

Figure 12. FPS experiment CPU-load figure from the project archive.

The results show that the FPS/percentage lever is one of the strongest software-level interventions for thermal management. The performance cost is substantial: average FPS falls from approximately 24.06 to 8.34. In return, GPU load decreases by approximately 46.5 percentage points and average temperature falls by 1.85 °C. The 2.19 °C decrease observed in the final ten-minute, near-steady-state interval is consistent with the delayed nature of thermal dynamics.

8.3 Resolution / imgsz Experiment

The purpose of this experiment was to quantify the effect of imgsz on temperature and to provide evidence for the FOPDT model parameters associated with the resolution control lever. The YOLO input size was reduced from 640 px to 320 px. This control lever lowers computational cost while directly trading against detection quality. In the experiment, FPS did not decrease; instead, average FPS increased by 2.12 because the smaller input size reduced inference time.

Phase	n	Duration	Mean FPS	Mean GPU load	Mean CPU load	Mean temperature	Min-Max temperature
640 px	3556	29.9 min	23.75	71.0%	19.0%	57.15 °C	47.44-58.53 °C
320 px	3571	30.1 min	25.87	67.0%	19.1%	55.99 °C	55.44-57.94 °C

Comparison	Mean difference (320 - 640)	Last 10 min difference
GPU temperature	-1.16 °C	-1.85 °C
FPS	+2.12	+2.09
GPU load	-4.0 points	-4.2 points
CPU load	+0.1 points	+0.2 points

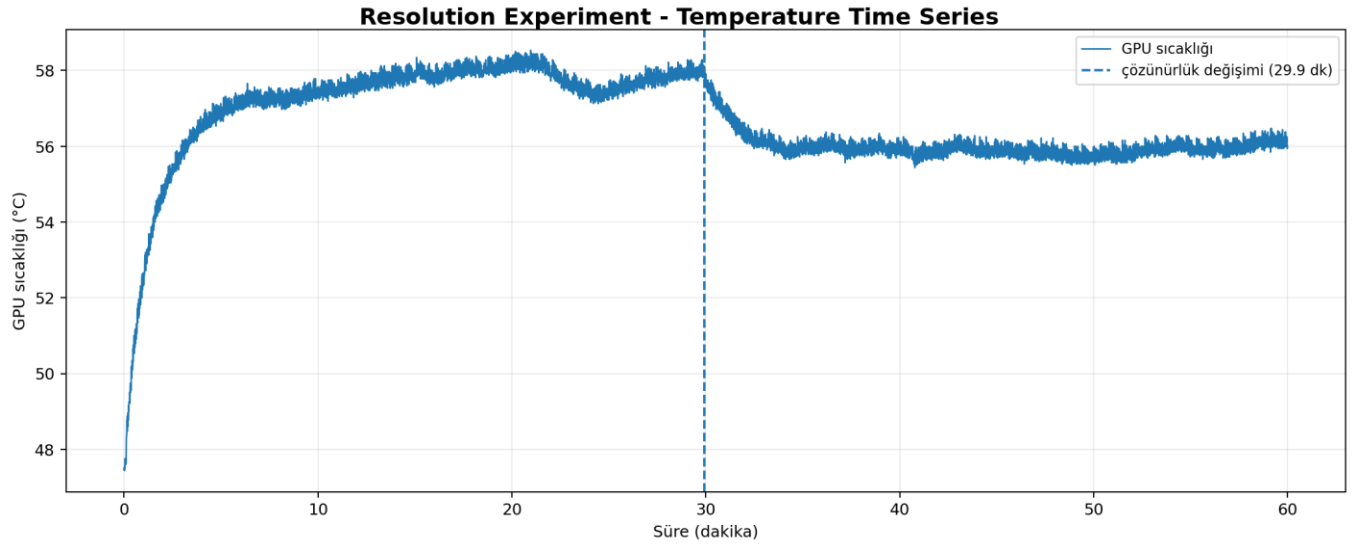

Figure 13. Resolution experiment: temperature time series after the transition from 640 px to 320 px.

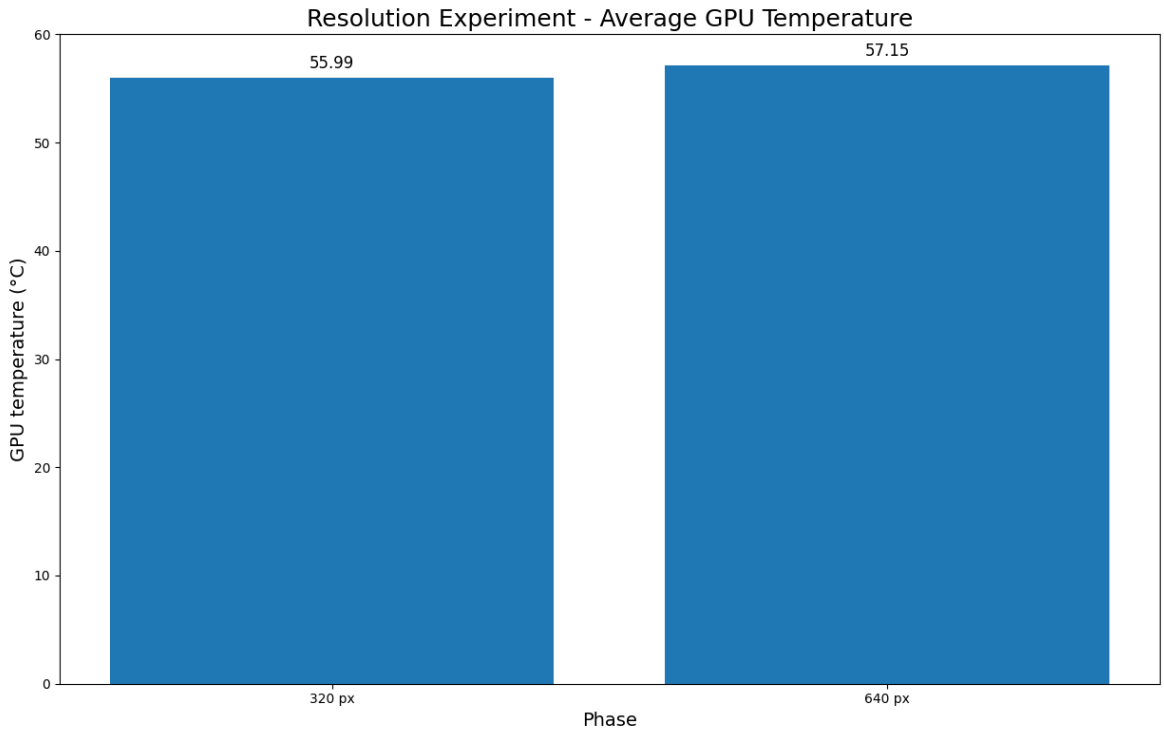

Figure 14. Resolution experiment: average GPU temperature by phase.

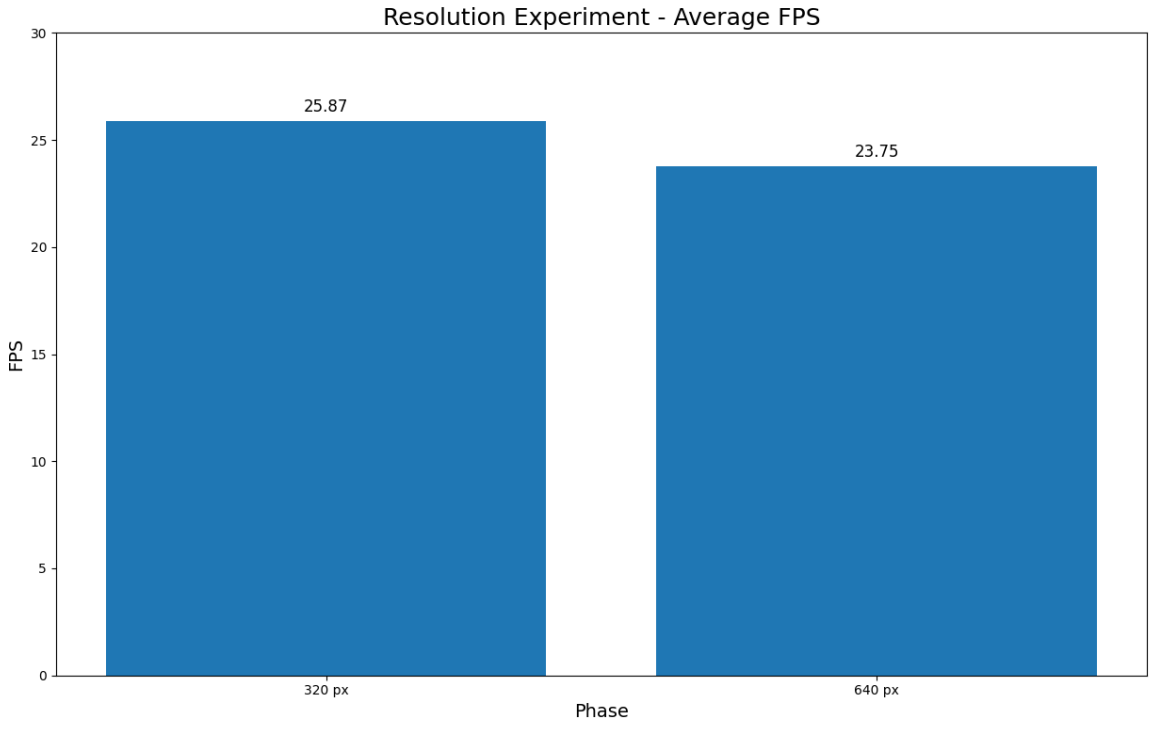

Figure 15. Resolution experiment: average FPS by phase.

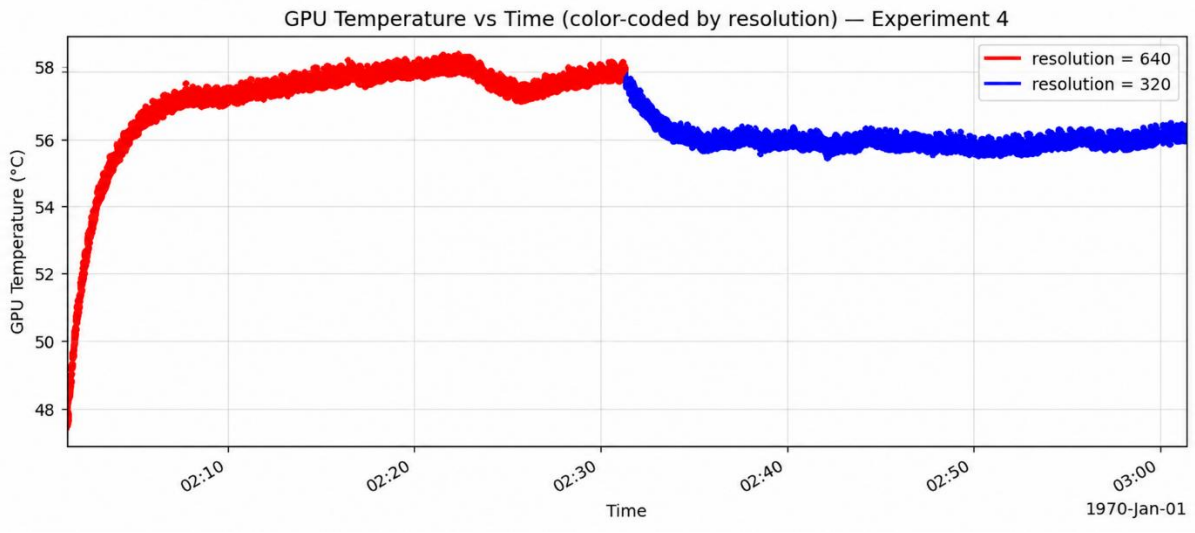

Figure 16. Resolution experiment temperature figure from the project archive.

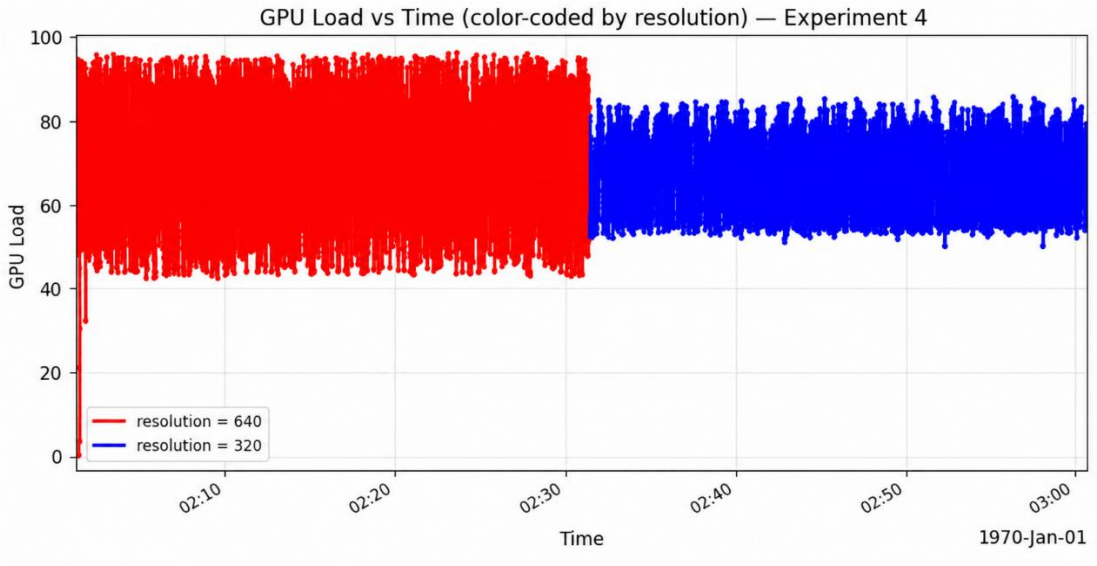

Figure 17. Resolution experiment GPU-load figure from the project archive.

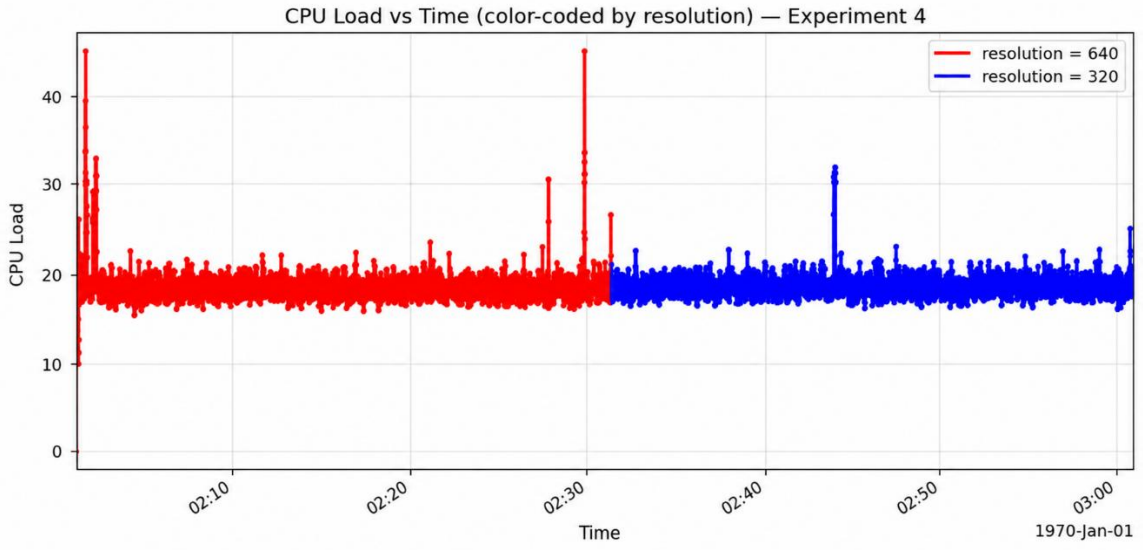

Figure 18. Resolution experiment CPU-load figure from the project archive.

Resolution reduction did not produce as aggressive a temperature decrease as FPS-percentage reduction; however, it is a practical control lever because it reduced temperature while preserving, and even increasing, FPS. Field deployments should also measure the resulting impact on detection quality. Therefore, a deployment-grade controller should consider not only temperature and FPS but also model accuracy or task success.

9. Discussion, Risks and Improvement Opportunities

9.1 Why a Fan Is Useful but Not Sufficient on Its Own

A fan and heatsink are the first line of defence for thermal management in edge AI devices. NVIDIA documentation also states that fan management is used in SoC thermal management to delay clock throttling to higher temperatures [W5]. However, a fan may not be sufficient in every operating condition: ambient temperature may be high, the device may operate in an enclosure, the fan profile may be quiet or conservative, dust may reduce airflow, or the model workload may exceed the cooling capacity. Because field reports still describe FPS loss, latency accumulation and instability even when fan or power settings are adjusted, an application-layer workload-management mechanism remains necessary [F4][F7].

9.2 System-Level Benefits

- System performance is degraded gradually and measurably instead of collapsing unpredictably.
- When the system enters the critical temperature band, inference resolution and processed-frame percentage are reduced to lower thermal load.
- Because decisions are made in a feedback loop, they are updated throughout operation according to temperature trend rather than applied as a one-off static setting.
- CSV telemetry makes it possible to audit which action was applied at which temperature during field operation.
- Reduced thermal stress can support longer-lived and more sustainable operation while decreasing the need for manual intervention.

9.3 Limitations

Limitation	Impact	Recommended improvement
Detection accuracy was not measured	The effect of lowering imgsz on mAP/recall is unknown.	Run joint temperature-FPS-accuracy experiments.
Fan speed and power/current telemetry are absent	The relationship between temperature change and power consumption is not fully visible.	Integrate tegrastats, INA3221 or Jetson power sensors.
Ambient temperature was not recorded	Experiments cannot be compared rigorously across environmental conditions.	Add ambient and enclosure-temperature sensors.
System clock was not synchronised in two experiments	Absolute time analysis is weakened.	Synchronise device time using NTP/chrony.
The fuzzy rule base is limited	Some transitions may remain coarse.	Compare with PID, MPC, RL or an optimised fuzzy rule base.
Emergency mode was not observed	Fail-safe behaviour was not stress-tested in the live experiment.	Run safe, controlled stress and recovery scenarios.

9.4 Future Work

- Optimise fan-control profiles and application-level throttling as a combined control problem.
- Compare model variants (YOLO n/s/m), quantisation and TensorRT optimisation within the same thermal-control framework.
- Run 8-24 hour long-duration experiments across multiple cameras and different lighting and ambient-temperature conditions.
- Extend the objective from temperature alone to a multi-objective target that includes latency, FPS, power consumption and detection accuracy.
- Compare the rule-based fuzzy system with a data-driven or adaptively tuned controller.
- Expose deployment profiles so users can select operating modes according to their stability, latency or quality requirements.
- Package the current open-source implementation as a Python library, for example through a pip install adaptive-edge distribution.

10. Conclusion and Engineering Assessment

The Adaptive Edge AI Controller is a working, experimentally supported research prototype for improving the thermal sustainability of long-running real-time inference on Jetson-class edge AI devices. Instead of passively waiting for kernel-level thermal throttling, it introduces controlled application-level workload reduction. This approach has practical value in security cameras, robotic systems, outdoor edge nodes and unmanned platforms where human intervention is limited.

The experiments show that both major control levers are effective. FPS/percentage reduction produces the strongest temperature decrease, while resolution/imgsz reduction can lower temperature while preserving or increasing FPS. The fact that the demonstration experiment did not enter the emergency band above 85 °C is a positive result; however, the 10.0% duration above 80 °C indicates that earlier intervention, fan/power telemetry and more advanced control parameters could further improve the system.

From an engineering perspective, the system is valuable because it is grounded in a real deployment problem, its codebase is modular, its telemetry enables repeatable analysis and its experimental results are supported by figures and tables. The main conclusion is that this work is not a substitute for physical cooling. It is an application-layer thermal-awareness component that complements hardware cooling and contributes to more stable, longer-lived and more sustainable edge AI operation.

Appendix A - Project Archive Inventory

Category	Files / Folders	Importance for the Report
Core package	thermal_edge/	Core control, sensor and telemetry code.
Demo	examples/basic_yolo_jetson.py	Real-time camera + YOLO + overlay integration.
Model	insan_tespit_mdeli.pt	YOLO model weights used for human detection.
Demo results	adaptive_edge_demo.csv, Sıcaklık-Değişim-Grafiği.png	Long-running closed-loop experiment and main temperature figure.
FPS experiments	fps_deney.csv, fps_deney.py, fps_deney*.png	Effect of the percentage/FPS control lever.
Resolution experiments	resolution_deney.csv, imgsz(resolution)_deney.py, resolution_deney*.png	Effect of the imgsz control lever.
Documentation	proje-detayları.docx, proje_analizi.md	Existing project descriptions and previous analysis notes.
Manual tests	tests/manual/*.py	Control-loop, preflight and FOPDT validation attempts.

Appendix B - CSV Schemas

CSV	Columns
adaptive_edge_demo.csv	timestamp, gpu_temp, gpu_load, cpu_load, fps, imgsz, percentage, temp_delta, mode
fps_deney.csv	timestamp, gpu_temp, resolution, percentage, actual_fps, gpu_load, cpu_load
resolution_deney.csv	timestamp, gpu_temperature, resolution, fps, gpu_load, cpu_load

References

The following sources were used to establish the problem definition, Jetson/edge AI context and field-reported user issues discussed in this report. All sources were reviewed and evaluated on 20 May 2026.

[W1] NVIDIA Jetson Orin Nano Developer Kit User Guide. The guide positions the Jetson Orin Nano Developer Kit as an entry-level development kit for AI robots, intelligent drones and smart cameras. https://developer.nvidia.com/embedded/learn/jetson-orin-nano-devkit-user-guide/index.html

[W2] NVIDIA Autonomous Machines / Jetson Platform. NVIDIA describes Jetson platforms as being used for robots, drones, intelligent video analytics and autonomous machines.https://www.nvidia.com/en-us/autonomous-machines/

[W3] NVIDIA Jetson Nano Product Development Page. Jetson Nano is positioned for running AI workloads such as classification, object detection, segmentation and speech processing within a low-power envelope. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/

[W4] NVIDIA Jetson Linux Developer Guide - Tegrastats Utility. The tegrastats utility reports memory, processor and related system statistics on Jetson-based devices. https://docs.nvidia.com/jetson/archives/r36.2/DeveloperGuide/AT/JetsonLinuxDevelopmentTools/TegrastatsUtility.html

[W5] NVIDIA Jetson Linux Developer Guide - Platform Power and Performance / Thermal Cooling. Jetson BSP provides thermal cooling through fan management and clock throttling; throttling can directly affect performance.https://docs.nvidia.com/jetson/archives/r36.5/DeveloperGuide/SD/PlatformPowerAndPerformance/JetsonOrinNanoSeriesJetsonOrinNxSeriesAndJetsonAgxOrinSeries.html

[F1] NVIDIA Developer Forums - YOLOv8 on Jetson Nano. A user running YOLOv8 on Jetson Nano reported core temperature rising to around 70 °C after roughly ten minutes, creating concern for outdoor autonomous use. https://forums.developer.nvidia.com/t/yolov8-on-jetson-nano/265650

[F2] NVIDIA Developer Forums - Overheating shut down - Jetson Nano. A user performing real-time object detection from an RTSP stream reported that the device shut down after 15-20 minutes and that the heatsink became very hot. https://forums.developer.nvidia.com/t/overheating-shut-down-jetson-nano/80693

[F3] Ultralytics Community - System Freeze When Performing Heavy YOLO Inferencing. A user example reports a system freeze after one to two hours under multi-camera YOLO inferencing, requiring manual restart. https://community.ultralytics.com/t/system-freeze-when-performing-heavy-yolo-inferencing/1882

[F4] NVIDIA Developer Forums - Delay due to nvinfer. A field example reports temperature and latency accumulation in a DeepStream pipeline, with drop-frame/interval settings reducing both temperature and latency. https://forums.developer.nvidia.com/t/delay-due-to-nvinfer/324632

[F5] GitHub dusty-nv/jetson-inference Issue #1473. A user issue reports very high CPU temperature and automatic shutdown while running a deep-learning model.https://github.com/dusty-nv/jetsoninference/issues/1473

[F6] NVIDIA Developer Forums - Jetson Nano Long run over heat. A Jetson Nano user raises concerns about heat during 24/7 OpenCV CNN-based face recognition.https://forums.developer.nvidia.com/t/jetson-nano-long-run-over-heat/157545

[F7] NVIDIA Developer Forums - Jetson Nano crashing while using detectnet-camera demo. A user reports crashes while running the detectnet-camera example; crashes decrease in 5 W mode, while FPS falls noticeably. https://forums.developer.nvidia.com/t/jetson-nano-crashing-while-using-detectnet-camera-demo-from-jetson-inference/74384

[F8] NVIDIA Developer Forums - Power error while using TensorFlow. A TensorFlow-based object-detection workload produced a "System throttled due to over-current" message alongside temperature increase. https://forums.developer.nvidia.com/t/power-error-while-using-tensorflow/181327

[F9] NVIDIA Developer Forums - Jetson Nano Freezes While Detecting. A user reports that a YOLO + OpenCV + Python system freezes after running for some time; tegrastats monitoring is suggested. https://forums.developer.nvidia.com/t/jetson-nano-freezes-while-detecting/214700

[F10] NVIDIA Developer Forums - DeepStream thermal throttle at 68-70 C. A Jetson Orin Nano Super user reports thermal throttling around 68-70 °C in a DeepStream pipeline. https://forums.developer.nvidia.com/t/deepstream-7-1-on-jetson-orin-nano-super-3-stream-pipeline-thermal-throttle-at-68-70-c-seeking-fps-optimization-advice/364742