Project Report

Sustainability Study on Video Transcodings

Group 7

TU Delft

Project Report

Sustainability Study on Video Transcodings

by

Roberto Negro (6302114),
Andrea Malnati (6293255),
Leonardo Lago (6317081)

Date: October 23, 2025
Course: Designing Sustainable ICT Systems
Responsible Lecturer: Przemysław Pawełczak

TU Delft

Contents

1 Experiment Design and JIT Optimization 1
1.1 Experiment Design . . . 1
1.2 JIT Optimization . . . 2

2 Environmental Impact Assessment 4
2.1 System Boundary . . . 4
2.2 Carbon Footprint Estimation . . . 5
2.3 Sensitivity Analysis . . . 6

3 Reflection 7
3.1 Limitations . . . 7
3.2 Final reflections . . . 7

A Supporting Material 9
A.1 Codec Comparison . . . 9
A.2 Google Argos VCU Comparison . . . 9

Bibliography 11

1

Experiment Design and JIT Optimization

Repository: `github.com/RobertoN0/DSICTS-team-7`

1.1. Experiment Design

Among the various components of video streaming pipelines, the transcoding stage is recognized as one of the most energy-intensive tasks [1]. For this reason, our study focuses exclusively on the daily workload of the transcoding process within a professional video server.
Based on this, we simulate the 24-hour equivalent workload of a *NETINT Quadra Video Server* [5], concentrating on its transcoding activity to estimate both average energy usage and the resulting carbon footprint under realistic operational conditions.
In our implementation, the server accepts an uploaded video and automatically produces a ladder of renditions: 1080p, 720p, 480p, and 360p, following the typical structure of adaptive bitrate streaming (ABR) workflows. We assume that the service operates in *real time*, with an average encoding speed greater than $1\times$.
We restrict our main analysis to the *H.264 codec*, given its prevalence in production environments and the availability of comparable energy data. For completeness, comparative results using *HEVC* and *AV1* are included in Appendix A.1.
In this experiment, we measured the energy consumption of concurrent transcoding requests executed with *GPU acceleration* rather than CPU encoding. This decision was due to technical issues in monitoring CPU-based *ffmpeg* processes, which prevented accurate attribution of energy consumption to the transcoding task. Conversely, GPU transcoding provides a clear separation between the server's CPU workload and the GPU power draw associated with the transcoding process. Further details on these measurement limitations and their implications are discussed in Sec 3.1.

The experiments were executed on an *ASUS Vivobook 16X* laptop equipped with an *Intel Core i9-13980HX (13th Gen)* CPU and an *NVIDIA RTX 4060 Mobile GPU*.

To execute our experimental workflow, we adopted the following procedure:

- Build the server executable
- Start the experiment loop:
  - Compile and start the server with the chosen JIT profile
  - Warm up hardware (only during the first iteration)
  - Wait for cooldown to restore optimal hardware temperature
  - Start monitoring processes (CPU and GPU)
  - Launch Locust to execute transcoding requests
  - Wait for the run to complete
  - Terminate the server and cooldown before next iteration
- Repeat for every JIT profile

To ensure consistency and comparability across runs, we kept the following parameters fixed:

- Number of repetitions: *30*
- Cooldown time between runs: *90 seconds*
- Input video: *1080p 30 fps*, fixed duration (3.15 min)
- Output resolutions: *1080p, 720p, 480p, 360p*
- Locust workload: *3 users*, spawn rate *1 request/s* (maintaining real-time encoding, speed > 1×)

1.2. JIT Optimization

In this section, we analyze the impact of Just-In-Time (JIT) compilation on our experiment. Since JIT optimizations directly affect the execution of Java bytecode, they influence only the behavior of our Spring Boot server, while the actual transcoding processes executed by ffmpeg run natively on the GPU and are therefore not impacted by JIT.

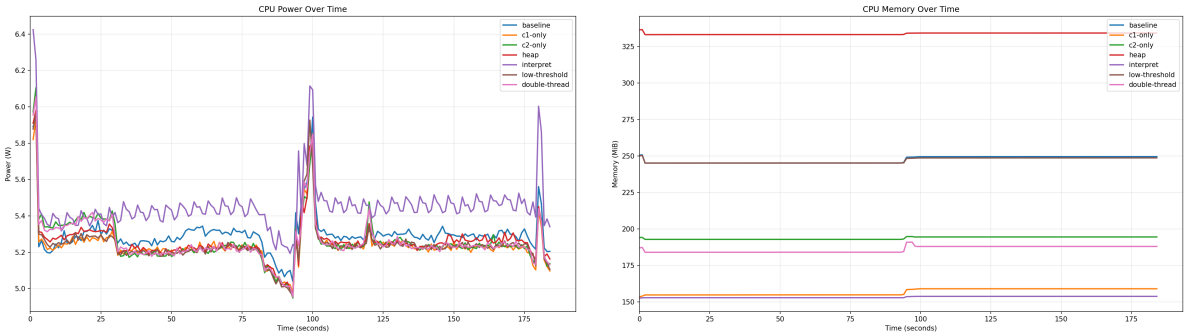
We applied seven different JIT profiles to the server and examined how their intended behaviors translate into variations in CPU performance, memory usage, and energy consumption. Because our service primarily acts as an orchestrator for GPU-accelerated encoders, the Java Virtual Machine (JVM) mainly contributes through request handling, command construction, and process management. Consequently, we expect relatively modest yet measurable differences among profiles, mostly arising from warm-up dynamics, compilation bursts, and memory residency patterns.

Profile	**Expected effect (warm–up / steady–state / memory & energy)**
baseline	C1 compiles quickly, hot code promoted to C2. Usually best overall time/energy for general services; moderate RSS.
interpret	No JIT. Lowest code-cache/RSS, but slower execution; longer wall time typically increases total energy.
c1-only	Fast warm–up, but never reaches C2 quality; good for very short runs, otherwise longer wall time and slightly higher energy.
c2-only	High-quality code after late warm-up; heavy compile bursts early. Beneficial only for workloads dominated by JVM activity; can hurt short or mixed Java/native tasks.
low-threshold	Compiles earlier/more methods; larger code cache and more compile activity up front; pays off only on long JVM-bound workloads.
double-thread	Enables two parallel JIT compiler threads; useful under many simultaneous hot spots; otherwise neutral to slightly higher contention.
heap	Larger fixed heap reduces GC pressure only if allocation-bound; otherwise inflates RSS with little time/energy benefit.

Table 1.1: JIT profiles and expected effects (qualitative).

Keeping these expectations in mind, the results confirm that JIT tuning has a limited influence on our workload, which is dominated by GPU-based transcoding rather than JVM execution. The power-over-time traces (Fig. 1.1a) show a clear stratification: *interpret* consistently exhibits the highest power draw, followed by *baseline*, while all other JIT profiles cluster closely below them with nearly identical average power. This behavior indicates that disabling JIT increases CPU effort during request handling, whereas tiered and optimized JIT modes converge to similar steady-state efficiency once the service is warmed up. The peak visible around 95 s correspond to the start of new request batch generated by Locust, rather than to JIT compilation activity.

Memory-over-time traces (Fig. 1.1b) show that *interpret* has the lowest memory usage, followed by *c1-only*. *double-thread* and *c2-only* stay in the middle range, while *baseline* and *low-threshold* use slightly more memory. As expected, *heap* allocates the most memory due to the larger heap size. Even though *low-threshold* compiles more methods earlier, it does not bring any real advantage in memory or energy, which confirms that this option is only useful for very long or JVM-heavy workloads.



(a) Power over time by profile.

(b) RSS over time by profile.

Figure 1.1: Temporal behavior of power and memory under different JIT profiles.

The aggregated metrics (Table 1.2) further support these observations. Total energy differences remain within 1–2%, with `baseline` and *c2-only* achieving slightly lower totals, and *interpret* the highest. Average CPU power varies by less than 0.2 W across all JIT profiles, and GPU-related metrics stay constant as expected. Overall, when the JVM primarily orchestrates native GPU processes, JIT configuration changes have minimal effect on system efficiency. The default tiered *baseline* remains the most balanced choice, while alternative profiles such as *c1-only* or *interpret* trade responsiveness for minor energy savings.

Metric	baseline	c1-only	c2-only	heap	interpret	low-threshold	double-thread
Total Energy (J)	7191.901	7171.639	7166.065	7185.079	7230.391	7167.172	7165.443
CPU Energy (J)	974.743	963.846	967.613	967.521	1003.185	964.283	967.434
GPU Energy (J)	6217.159	6207.793	6198.452	6217.557	6227.206	6202.889	6198.010
Avg Total Power (W)	33.554	33.356	33.396	33.492	33.622	33.386	33.403
Avg CPU Power (W)	5.269	5.210	5.230	5.230	5.423	5.212	5.229
Avg GPU Power (W)	28.286	28.146	28.166	28.263	28.199	28.173	28.174
Avg CPU Mem (MiB)	247.338	156.817	193.669	333.780	153.304	246.944	186.045
Avg GPU Mem (MiB)	2518.653	2506.748	2518.180	2506.772	2566.722	2506.446	2516.967

Figure 1.2: Summary table of different JIT profiles for H.264 experiment run

2

Environmental Impact Assessment

2.1. System Boundary

This environmental impact assessment focuses on a representative NETINT server used for video transcoding and streaming, specifically the *Quadra Video Server*. The system is equipped with an x86 CPU (matching the architecture used in our experiments) and ten *T1A Video Processing Units* (VPUs). VPUs are Application-Specific Integrated Circuits (ASICs) designed to perform video processing tasks, such as transcoding, with high efficiency. We also provide a comparison with Google's Argos VCU to validate our results; since this is not the main focus of our assessment, it is included in Appendix A.2.

Our analysis estimates the environmental impact of a single Quadra server operating in the Netherlands over a 24-hour period. The embodied carbon of the hardware is excluded due to limited data availability, particularly for ASIC components and server manufacturing.

To estimate energy use, we first measure the power consumption required for real-time transcoding of 1080p, 30 fps video on an Nvidia RTX GeForce 4060 Mobile GPU, and then compare it to the performance of a NETINT T1A VPU to obtain a relative efficiency ratio [4].

Since the 4060 Mobile is a consumer-grade GPU, Nvidia imposes limits on the number of concurrent NVENC sessions. We used a publicly available patch from GitHub to lift this restriction [3]. We then executed a simple Bash script launching 11 parallel FFmpeg processes to transcode 1080p@30 fps videos, in order to reach the maximum GPU workload that still maintained real-time speed (>1×). This was necessary to match the reference data for power per stream, which are measured under full-load conditions.
For H.264 transcoding, the measured values were:

$$P = 29.455\ W \quad \text{Speed} = 1.075$$

which yields a per-stream power consumption of:

$$\frac{P}{N \cdot \text{Speed}} = \frac{29.455 W}{11 \cdot 1.075} = 2.491 \quad W/\text{stream}$$

To validate this estimate, we compared it with data for the Nvidia Tesla T4:

$$\frac{P}{\text{Streams}} = \frac{70}{10.7} = 6.542 \quad W/\text{stream}$$

This comparison is reasonable, as the T4 employs an older NVENC version (7.0) compared to the 8.1 encoder used in the 4060 Mobile, and it typically performs worse in benchmark tests.

For the NETINT T1A VPU, reported data show:

$$\frac{P}{\text{Streams}} == \frac{20}{32} = 0.625 \quad W/\text{stream}$$

Comparing the two devices gives the relative efficiency:

$$\frac{\text{T1A}}{\text{RTX 4060}} = \frac{0.625}{2.491} = 0.251$$

During our experiments, the GPU consumed an average of 6217.159 J of energy. Using the efficiency ratio, a single VPU would therefore consume 1560.507 J over three minutes, and a full server with ten VPUs would consume 15 605.07 J under full load. Over a three-hour period, this corresponds to 936 304.2 J.

2.2. Carbon Footprint Estimation

To model a full day of server operation, we constructed an estimated workload distribution, shown in Table 2.1. This distribution assumes varying transcoding demand throughout the day, with higher activity during evening hours.

Time	Workload	Carbon intensity factor
00:00 - 03:00	45%	394.667
03:00 - 06:00	10%	308.667
06:00 - 09:00	10%	292.0
09:00 - 12:00	20%	246.333
12:00 - 15:00	30%	276.333
15:00 - 18:00	50%	480.0
18:00 - 21:00	70%	465.667
21:00 - 00:00	60%	422.667

Table 2.1: Distribution of server workload during the day

We chose this distribution because content creators may prefer uploading during the evening or night so that by the next day, content is fully processed and available to users. Meanwhile, many streaming platforms and creators schedule live content in the evening to align with higher audience availability. For example, Johnson et al. [2] studied Twitch streamers' scheduling decisions and found that many choose times with higher audience presence.

Carbon intensity data for the Netherlands were obtained from Nowtricity [6] at this date 2025-10-19-15:00. The data collected is also shown in Table 2.1. The total daily emissions for transcoding activities are computed as:

$$\text{Daily Carbon emission} = E_{3h} \sum_{i=0}^{T} L_i C_i$$

where T is the number of time intervals, L_i is the relative workload and C_i is the carbon intensity factor for interval i.

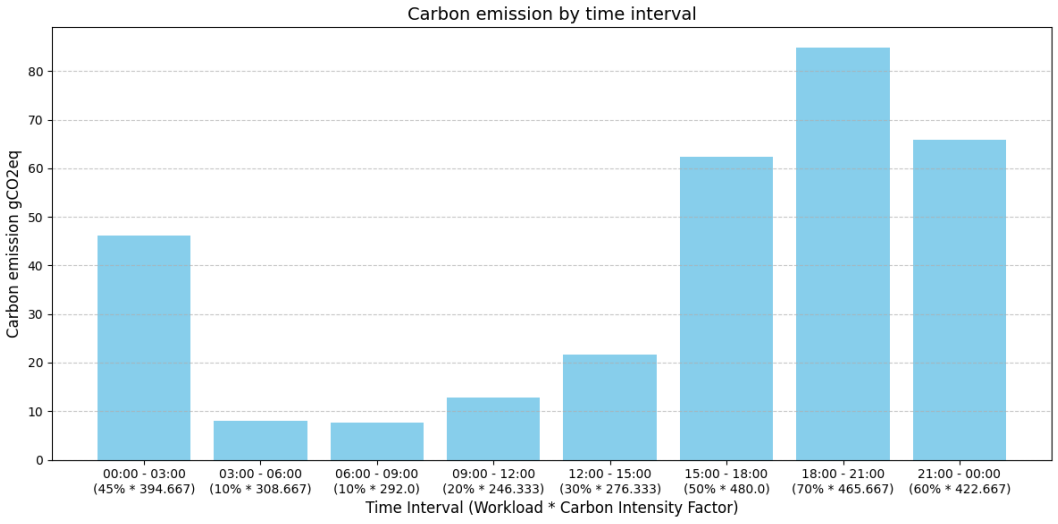


Figure 2.1: Carbon emission by time interval

Based on this model, the total daily emissions are 309.345 gCO_2eq, corresponding to approximately 12.89 gCO_2eq/h for transcoding operations. Figure 2.1 shows the carbon footprint using our model.

We compared this value with data reported by Afzal et al. [1], who estimate an average of 76 gCO_2eq/h for streaming in Germany, across mixed resolutions and considering the full streaming pipeline. Our estimate, limited to the transcoding stage, shows significantly lower emissions, which is consistent with expectations. Modern VPUs are approximately four times more power-efficient than traditional CPU/GPU-based transcoding hardware, and regional differences in carbon intensity also contribute. This comparison is purely quantitative and intended to provide a consistency check, helping to verify that our measurement and calculation are within a plausible range. A more detailed comparison is difficult due to the specific limitations of our experiment and the assumptions made. These aspects are discussed in more detail in the following sections.

2.3. Sensitivity Analysis

In this section, we summarize and evaluate the main assumptions made during the design of the experiment and the environmental impact assessment. Each assumption introduces some degree of uncertainty that may affect the accuracy of the final energy and carbon footprint estimates.

Transcoding and performance assumptions. We assumed that all requests correspond to 1080p videos at 30 fps, as this matches the configuration used in the NETINT reference data for the T1A VPU, although with current technology this represents a conservative estimate compared to today's typical content. We also assumed that every encoding operation sustains real-time speed (encoding speed > 1×). As only H.264 encoding was considered, differences for HEVC and AV1, discussed in Appendix A.1, may slightly change both energy and carbon footprint estimates.

Hardware proportionality assumptions. Our proportional analysis is based on power-per-stream data reported by NETINT for the T1A VPUs, which we assume to be accurate and representative of real operating conditions. Any deviation in these reference values would directly affect the estimated energy and carbon footprint of the simulated Quadra server, although the relative comparison across profiles would remain valid.

Environmental and operational assumptions. For the environmental impact estimation, we assumed that the server operates in the Netherlands and applied the corresponding average carbon intensity factors. We considered these factors constant over time, neglecting short-term weather or seasonal variations. In reality, carbon intensity in the Netherlands can vary considerably, for instance, average values in March and May were 512 and 252 gCO_2eq/kWh, respectively, which are 42% higher and 30% lower than the factor used in our analysis. We also assumed that the upload workload pattern described earlier reflects a realistic daily distribution. This impacts the total emissions, as our low-workload intervals coincide with periods of lower carbon intensity. For example, shifting the workload from 15:00–00:00 to 06:00–15:00 would yield approximately a 20% reduction in total emissions.

3

Reflection

3.1. Limitations

In this section we go in detail on what factors limited our experiment design and results.

- Design and implementation limitations
  - JIT impacts: our server's main employment regards running `ffmpeg` commands as sub-processes. Since these commands are not Java related, the use of different JIT optimizations brings little or negligible improvements to the general energy consumption.
  - Monitoring of CPU ffmpeg commands: our initial plan included monitoring the energy consumed by CPU-only experiments, without GPU acceleration. However, this turned out to be more difficult than expected, as we were unable to accurately track the energy used by the multiple `ffmpeg` processes launched by the server. This limitation comes from the monitoring script, which only tracks the main server process and not the dynamic sub-processes created when new transcoding requests arrive. Ideally, the monitor should start dynamically when each new process is spawned, but implementing such tracking proved unsuccessful. As a result, we decided to focus on experiments where the server runs on CPU but the transcoding is executed on the GPU, since the GPU monitoring script can capture the total power draw of all `ffmpeg` processes running simultaneously. This approach allowed us to obtain clear and consistent energy measurements.
- Results limitations
  - Our final results on total energy and carbon footprint are highly dependent on the assumptions made throughout the analysis. In particular, the proportional scaling between our GPU and the NETINT VPUs was based on manufacturer data, which introduces uncertainty as the real efficiency of specialized hardware may differ.
  - Another limitation concerns the hardware used in our experiments. Although we successfully removed the NVENC session limit from the RTX 4060 Mobile, we cannot fully ensure that no hidden power or performance capping remained active. This prevented us from testing heavier workloads, so we limited the number of concurrent transcoding processes to maintain an average `ffmpeg` speed above 1×.

3.2. Final reflections

After having run and analyzed the working and consumption of a transcoding server, we propose the following reflections. The various compiling profiles offer little difference when analyzed, so there is not much to say in regard. For the overall Carbon Footprint instead, our results are dependent on Carbon Intensity Factor, which changes every hour, and also the considered location, as every Nation has its values. The impact on sustainability also varies based on the time of the year and weather of the day. For this reason it is hard to establish measurable optimizations that would help sistematically reduce the Carbon Footprint of such server. Nevertheless we here propose an idea that could help in this.

We consider a server that is able to perform transcoding in-real-time, which can be of great use for cases like Twitch lives, but it also represents a possible YouTube server. For this last case, we have thought about a scenario where a youtuber is uploading a video to his channel in advance with respect to when he wants it published. For such case we then propose a method that takes into account the Carbon Footprint by considering the Carbon Intensity Factor of the server's location at the upload hour. In fact, one way to reduce the Carbon Footprint is to shift the workload of the server in a way that the server consumes more when the energy is the cleanest. For this reason, if the platform that is ingesting the video can take this in consideration, it could also schedule the workload in a way that it pollutes less, giving the uploader the option to select a *sustainable upload* mode that delays processing to a cleaner energy window.

A

Supporting Material

A.1. Codec Comparison

Using the same setup described in Section 1, we repeated the experiment with the *HEVC (H.265)* and *AV1* codecs under the *baseline* profile and identical transcoding parameters. The results in Figure A.1 show that H.264 and AV1 consume similar GPU energy, while HEVC requires fewer Joules overall, despite a longer transcoding duration.

This confirms that the HEVC NVENC module is more energy-efficient, as it processes less data per frame and maintains lower average power, even if its encoding complexity increases runtime. In contrast, AV1 does not yet provide measurable efficiency gains on current NVIDIA GPUs.

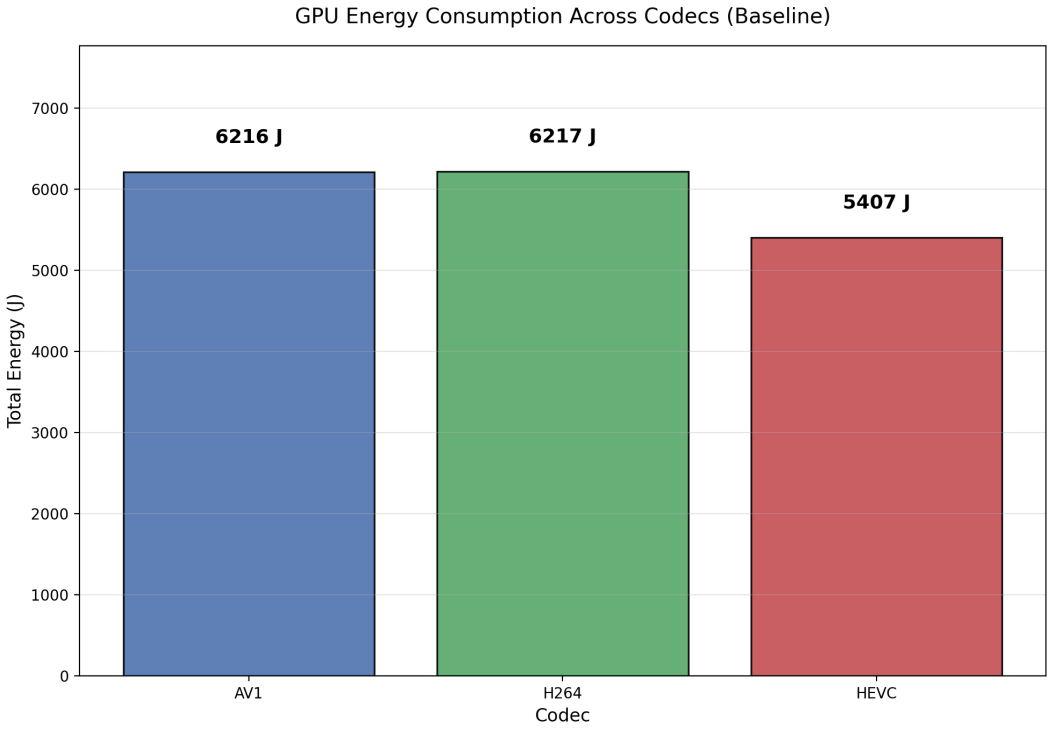


Figure A.1: GPU energy consumption across codecs.

A.2. Google Argos VCU Comparison

We compare the results obtained in our experiments with the data reported by Google [7]. In their work, the transition from CPUs to their custom ASIC, the Argos Video Coding Unit (VCU), achieved a 6.7× performance improvement for single-output H.264 transcoding.

Our test system, equipped with an Intel Core i9-13980HX (13th Gen) CPU, is capable of transcoding 22 simultaneous 1080p30 video streams in real time. This corresponds to an approximate throughput of 1368 Mpix/s, computed as follows:

$$\text{Throughput} = width \cdot height \cdot fps \cdot streams = 1080 \cdot 1920 \cdot 30 \cdot 22 = 1368576000 \text{ pix/s}$$

This throughput is approximately 1.91x higher than that achieved by the Skylake CPUs previously used by Google. This improvement is largely attributable to the six-year gap between the two hardware generations.

Each software-based transcoding process using ffmpeg consumes about 7.5 W. Therefore, the performance-per-watt of our CPU is given by:

$$\frac{perf_{CPU}}{P_{CPU}} = \frac{1368.576000}{7.5 \cdot 22} = 8.2944 \, \frac{\text{Mpix/s}}{W}$$

Google does not directly report the power consumption of their VCU. However, assuming that its power requirements are roughly the same, or at least comparable, to those of the NETINT VPU, we can estimate the same efficiency metric for the VCU as:

$$\frac{perf_{VCU}}{P_{VCU}} = \frac{14932}{20 \cdot 20} = 37.33 \, \frac{\text{Mpix/s}}{W}$$

Comparing the two results shows that the Argos VCU is approximately 4.5× more power-efficient than our CPU for single-output encoding. This is somewhat lower than Google's reported 6.7× improvement, but the difference is reasonable given that our CPU is based on significantly newer hardware.

For completeness, we also compare our CPU with the NETINT T1U VPU. Since we measured the power consumption per ffmpeg process (as tracking all concurrent processes is impractical), we assume the same value for each transcoding stream:

$$\frac{P}{\text{Streams}} = 7.5 \quad W/\text{stream}$$

Thus, the ratio between CPU and VPU power consumption is

$$\frac{\text{VPU}}{\text{CPU}} = \frac{0.625}{7.5} = 0.0833$$

This indicates that the NETINT T1U VPU is roughly 12× more power-efficient than our CPU.

Finally, comparing Google's VCU and the NETINT VPU shows that the latter is approximately 2.6× more efficient. However, this comparison should be interpreted with caution, as Google's power consumption data are not publicly available, making our estimates highly sensitive and potentially imprecise.

Bibliography

[1] Samira Afzal, Narges Mehran, Zoha Azimi Ourimi, Farzad Tashtarian, Hadi Amirpour, Radu Prodan, and Christian Timmerer. A survey on energy consumption and environmental impact of video streaming, 2024. URL https://arxiv.org/abs/2401.09854.

[2] Mark R Johnson. Time zones, time slots and twitch: When do game streamers go live, and why? *New Media & Society*, 2025. doi: 10.1177/14614448251381718. URL https://doi.org/10.1177/14614448251381718.

[3] keylase. nvidia-patch. URL https://github.com/keylase/nvidia-patch. Accessed: 2025-10-19.

[4] NETINT Technologies. Choosing an encoder. https://netint.com/choosing_an_encoder/, September 2022. Accessed October 20, 2025.

[5] NETINT Technologies. Quadra video server. https://netint.com/products/quadra-video-server/, September 2022. Accessed October 20, 2025.

[6] Nowtricity. Nowtricity. URL https://www.nowtricity.com/. Accessed: 2025-10-19.

[7] Parthasarathy Ranganathan, Daniel Stodolsky, Jeff Calow, Jeremy Dorfman, Marisabel Guevara, Clinton Wills Smullen IV, Aki Kuusela, Raghu Balasubramanian, Sandeep Bhatia, Prakash Chauhan, Anna Cheung, In Suk Chong, Niranjani Dasharathi, Jia Feng, Brian Fosco, Samuel Foss, Ben Gelb, Sara J. Gwin, Yoshiaki Hase, Da-ke He, C. Richard Ho, Roy W. Huffman Jr., Elisha Indupalli, Indira Jayaram, Poonacha Kongetira, Cho Mon Kyaw, Aaron Laursen, Yuan Li, Fong Lou, Kyle A. Lucke, JP Maaninen, Ramon Macias, Maire Mahony, David Alexander Munday, Srikanth Muroor, Narayana Penukonda, Eric Perkins-Argueta, Devin Persaud, Alex Ramirez, Ville-Mikko Rautio, Yolanda Ripley, Amir Salek, Sathish Sekar, Sergey N. Sokolov, Rob Springer, Don Stark, Mercedes Tan, Mark S. Wachsler, Andrew C. Walton, David A. Wickeraad, Alvin Wijaya, and Hon Kwan Wu. Warehouse-scale video acceleration: co-design and deployment in the wild. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 600–615, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446723. URL https://doi.org/10.1145/3445814.3446723.