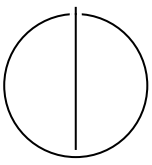
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Towards Edge Power Measurement: A Low-Cost, Programmable Power Meter

Chang Lu



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

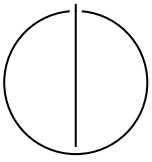
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Towards Edge Power Measurement: A Low-Cost, Programmable Power Meter

Leistungsmessung für Edge-Geräte: ein kostengünstiges, programmierbares Leistungsmessgerät

Author:	Chang Lu
Examiner:	Prof. Dr.-Ing. Jörg Ott
Supervisor:	Wei Geng
Submission Date:	31.08.2026



I confirm that this master's thesis is my own work and I have documented all sources and material used.

During the preparation of this work, AI assistants were used: Claude (Anthropic) assisted with code and text improvement, and ChatGPT (OpenAI) was used for helping with searching and understanding concepts. All AI-assisted content was reviewed and verified by the author, who takes full responsibility for this thesis.

Munich, 31.08.2026 Chang Lu

Acknowledgments

I would like to thank my supervisor, Wei Geng, for his guidance throughout this thesis, from the first hardware prototype to the final text. His detailed feedback in our regular discussions shaped both the platform and this document.

I also would like to thank Prof. Dr.-Ing. Jörg Ott and the chair for providing the reference instrument and the edge devices used in the experiments.

Finally, I thank my family and my friends for their support during my studies.

Abstract

Many edge devices cannot measure their own power consumption, and devices with onboard sensors miss peripheral power consumption usage, such as NICs which is sometimes critical for power-constrained scenarios. For example, an offloading experiment, power consumption can be better measured by a non-intrusive power meter. Laboratory instruments are accurate, but they are expensive and not practical for a testbed with several devices. Cheaper open-hardware meters often lack wireless access and programmable APIs.

This thesis addresses this gap by constructing a networked power meter. It utilizes an INA current sensor and an ESP32-C6 microcontroller, costing approximately €27. The firmware provides mutually exclusive acquisition modes: averaged monitoring (1-50 Hz), burst and streaming capture (100-3000 Hz configured), and transition capture (2 kHz practical; 2.4 kHz absolute limit) for short-duration load events. All data is transmitted via WiFi through MQTT or an HTTP RESTful API.

Evaluations included measurement accuracy, time synchronization, and transmission performance. After calibration, the current error remained within 0.10% on the Pi 5 and Jetson channels in independent validation runs. Cross-correlation analysis of natural load transients aligned the power trajectory with the host logs, with a median error of 132 µs—less than the sampling interval at 2 kHz—without requiring any modifications to the device. A message takes only 0.72 ms, allowing multiple consumers to subscribe simultaneously.

We conducted offloading case studies using this meter on an ESP32-S3, a Raspberry Pi 5, and a Jetson Orin Nano. The cost of a single ResNet-8 inference on a Pi 5 is 0.55 mJ, while on an ESP32-S3 it is 173.2 mJ, and the fixed cost per round is lower than the cost of a single local inference. Under this baseline accounting, and with the communication energy not included, offloading therefore saves energy from the first inference of a round.

Zusammenfassung

Viele Edge-Geräte können ihren eigenen Stromverbrauch nicht messen, und Geräte mit integrierten Sensoren erfassen den Verbrauch von Peripheriekomponenten – wie etwa Netzwerkkarten (NICs) – nicht; dies ist in Szenarien mit begrenzter Energieversorgung bisweilen entscheidend. Bei einem Offloading-Experiment lässt sich der Stromverbrauch beispielsweise präziser mit einem nicht-invasiven Leistungsmessgerät erfassen. Laborgeräte sind zwar genau, aber teuer und für Testumgebungen mit mehreren Geräten unpraktisch. Günstigeren Messgeräten auf Open-Hardware-Basis fehlen hingegen oft drahtlose Schnittstellen und programmierbare APIs.

Diese Arbeit schließt diese Lücke durch die Entwicklung eines vernetzten Strommessgeräts. Es verwendet einen INA-Stromsensor sowie einen ESP32-C6-Mikrocontroller und verursacht Kosten von etwa 27 €. Die Firmware bietet sich gegenseitig ausschließende Erfassungsmodi: die Überwachung von Durchschnittswerten (1–50 Hz), die gepufferte und kontinuierliche Erfassung (100–3000 Hz konfiguriert) sowie die Erfassung von Lastwechseln (praktisch 2 kHz; maximal 2,4 kHz) für kurzzeitige Lastereignisse. Die Datenübertragung erfolgt per WLAN mittels MQTT oder einer HTTP-REST-API.

Die Evaluierungen umfassten Messgenauigkeit, zeitliche Ausrichtung und Übertragungsleistung. Nach der Kalibrierung blieb der Stromfehler auf den Pi-5- und Jetson-Kanälen in unabhängigen Validierungsläufen innerhalb von 0,10 %. Die Kreuzkorrelationsanalyse natürlicher Lasttransienten ermöglichte die Abstimmung des Leistungsverlaufs mit den Host-Protokollen. Der mittlere Fehler betrug 132 µs – weniger als das Abtastintervall von 2 kHz – ohne dass Änderungen am Gerät erforderlich waren. Eine Nachricht benötigt nur 0,72 ms, sodass mehrere Nutzer gleichzeitig abonnieren können.

Wir führten Offloading-Fallstudien mit diesem Messgerät auf einem ESP32-S3, einem Raspberry Pi 5 und einem Jetson Orin Nano durch. Die Kosten einer einzelnen ResNet-8-Inferenz betragen auf einem Pi 5 0,55 mJ, auf einem ESP32-S3 hingegen 173,2 mJ. Die Fixkosten pro Runde sind dabei niedriger als die Kosten einer einzelnen lokalen Inferenz. Bei dieser Baseline-Rechnung und ohne Berücksichtigung der Kommunikationsenergie spart das Auslagern daher bereits ab der ersten Inferenz einer Runde Energie.

Contents

1. Introduction

1.1. Motivation

The edge computing puts computation near the place where data is produced. When a device cannot execute a task within its latency or energy budget, it can *offload* the task to a more capable device. Recent systems make this decision at run time: SMOOTH schedules several vision tasks on one shared backbone [11], budget-adaptive routing decides when the strong model is needed [10], and KUT probes the network path before it places a task [9]. For choosing a target, accuracy and latency can be obtained in software. Energy is the troublesome part. It needs a physical measurement at the device power input, and this value is often absent.

Many edge devices provide no power measurement at all. Those that do—through onboard sensors such as `tegrastats` on NVIDIA Jetson modules or the power-management integrated circuit (PMIC) on the Raspberry Pi 5—report only a fraction of the actual input power. Shalavi et al. [21] find that Jetson onboard sensors under-report by up to 50%. Yang et al. [30] show that `nvidia-smi` on data-center GPUs updates its reading during only 25% of the runtime, causing energy estimates to err by up to 70%.

External measurement instruments can fill this gap. In a multi-device edge deployment, however, the available choices soon become awkward. Research-grade instruments such as the Monsoon High Voltage Power Monitor (HVPM) ($1107, one main channel, 13.5 V maximum) and the Otii Ace Pro ($1699/channel) are expensive to scale to multiple devices. Open-hardware platforms such as RocketLogger [23] and Shepherd Nova [8] target low-voltage Internet of Things (IoT) nodes (4.8 V to 5.5 V) and do not cover the 19 V supply of GPU compute modules. No existing instrument combines multi-channel measurement, a 5 V to 19 V input range, network-accessible data, traceable calibration, and a parts cost below €50.

1.2. Design Goals

The goal of this work is a measurement platform that answers three practical questions. Can a low-cost, networked power meter reach an accuracy that is sufficient for offloading research, across the 5 V to 19 V supplies of common edge devices? How should the

data acquisition and the network transport be designed, so that continuous monitoring and the capture of millisecond-level events can be provided as mutually exclusive modes on one small device? And when the power meter and the device under test run on independent, unsynchronized clocks, how precisely can the measured energy be attributed to individual tasks?

The rest of the thesis keeps returning to these questions: the design chapter derives its requirements from them, and the evaluation answers each of them with a measured limit.

1.3. Contributions

The main outcome is a complete measurement path, from the supply cable to the task-energy value used by an offloading experiment. Its contributions are summarized below.

1. **A three-channel power meter with traceable calibration**. The platform uses INA219, INA226, and INA228 current sensors on an ESP32-C6 microcontroller, covers supply voltages of 5 V to 19 V, and costs approximately €27 in parts. After a two-parameter linear calibration against the Monsoon HVPM, the independently verified current residual is at most 0.10% on the Jetson and Pi 5 channels (worst case 1.7% on the auxiliary INA228 channel, at operating points where the reference itself is uncertain by about 1%).

2. **Mutually exclusive acquisition modes with MQTT transport.** Averaged monitoring at 1 Hz to 50 Hz serves continuous observation; transition capture at a practical 2 kHz (absolute limit: 2.4 kHz) records individual events with sub-millisecond boundary placement. Message Queuing Telemetry Transport (MQTT) delivers each monitoring message in 0.72 ms—30 to 117 times faster than Hypertext Transfer Protocol (HTTP) POST—and scales to eight concurrent subscribers without measurable device-side cost.

3. **Post-hoc time synchronization without hardware modifications**. A cross-correlation method aligns the power trace to the host's event log using natural load transitions as markers. The median alignment residual is 132 µs, contributing less than 0.19% to the per-event energy error on workloads of approximately 0.2 s to 1.4 s. No synchronization infrastructure, no device under test (DUT) firmware changes, and no GPIO connections are required.

A case study later in the thesis applies the complete platform to the offloading question. It measures the Pi 5's marginal cost at 0.55 mJ per inference—roughly

300 times below the ESP32-S3—bounds the per-round fixed overhead below 0.08 J, and shows that under this baseline convention offloading saves energy from the first inference of a round (communication cost excluded).

1.4. Thesis Outline

The background in Chapter 2 first introduces the measurement and communication concepts used throughout the work. This is followed by the related instruments and time-synchronization approaches in Chapter 3. The main part starts with the system design in Chapter 4. Chapter 5 then goes into the firmware and server, including several details which became important only during the experiments.

The evaluation is given in Chapter 6; calibration and sampling behavior take more space there because they determined the usable operating points of the power meter. After that, Chapter 7 uses the measured values for a small offloading comparison. Finally the results and the still open work are discussed in Chapter 8.

2. Background

Power is easy to name but less easy to measure at the boundary of an edge device. Before presenting our power meter, some background is needed on what the shunt actually measures and on how its samples travel to an experiment. We also introduce clock synchronization and the inference workloads here. These topics are not completely separate; the sensor aperture, network protocol and task duration later determine the same energy result.

2.1. Edge Computing and Task Offloading

The edge computing places computation on devices which are physically close to data source, instead of putting every operation in a remote cloud [22]. In a typical deployment, sensor nodes, microcontrollers, or single-board computers acquire data and run local processing, while a nearby gateway or server provides additional compute capacity. The obvious advantage is latency. Data does not need to travel to a remote data center before a result is available.

Task offloading. When a device cannot complete a task within its latency, accuracy, or energy budget, it can *offload* the task to a more capable device. The decision normally considers task accuracy and latency together with consumed energy. Software can measure accuracy and latency. Energy, however, requires a physical measurement at the device power input—onboard sensors, where they exist, report only a fraction of the total input power, as discussed in the next chapter.

Why energy matters for offloading. If the offloading platform consumes more energy to transmit a task and receive result than executing it locally, this means offloading can save the time but still waste energy. The break-even point depends on the communication cost, the local execution cost, and the ratio of batched tasks per communication round. That is why the comparison has to be made on real hardware. A timing result alone is not sufficient.

2.2. Power Measurement with Shunt Resistors

A shunt resistor is a low-value, precision resistor placed in series with the load. By Ohm's law, the voltage across the shunt is proportional to the current through it:

$$I = \frac{V_{\text{shunt}}}{R_{\text{shunt}}} \tag{2.1}$$

An analog-to-digital converter (analog-to-digital converter (ADC)) samples V_{shunt} and computes the current. The bus voltage V_{bus} is measured on a second ADC channel, and the instantaneous power follows as $P = V_{\text{bus}} \times I$. Energy over a time interval $[t_0, t_1]$ is the integral of power:

$$E = \int_{t_0}^{t_1} V_{\text{bus}}(t) \cdot I(t)\, \mathrm{d}t \tag{2.2}$$

In a sampled system, this integral is approximated by a trapezoidal sum over discrete samples.

High-side vs. low-side sensing. The shunt can be placed between the supply and the load (*high-side*) or between the load and ground (*low-side*). High-side sensing preserves the load's ground reference, which avoids ground-bounce errors on digital buses, but requires the ADC to reject a high common-mode voltage. All three sensors used in this work support high-side sensing.

Burden voltage. The voltage drop across the shunt reduces the voltage available to the load. For a $0.100\,\Omega$ shunt at 100 mA, the burden is 10 mV—negligible relative to a 5 V supply. For a $0.015\,\Omega$ shunt at 2 A, the burden is 30 mV. At the same current, choosing a smaller shunt value reduces the burden but also reduces the signal-to-noise ratio at the ADC input.

Calibration. Manufacturing tolerances in the shunt resistor and gain errors in the ADC cause systematic deviations. A two-parameter linear calibration is used to improve the accuracy:

$$I_{\text{corrected}} = \frac{I_{\text{raw}} - I_{\text{offset}}}{g} \tag{2.3}$$

where g is the gain and I_{offset} is the offset, both determined by least-squares regression against a reference instrument. This simple model is standard practice for current-sensing instruments [23].

2.3. The INA Sensor Family

Texas Instruments manufactures a family of digital power monitor integrated circuits under the *INA* designation. Each device integrates a shunt voltage ADC, a bus voltage ADC, and a digital interface (I^2C or SPI). We use the INA219, INA226 and INA228 from this family. Their key parameters are collected in Table 2.1.

Model	Shunt resolution	Bus V max	Conversion time (min)	Energy accumulator
INA219	12-bit (10 µV/LSB)	26 V	532 µs	none
INA226	16-bit (2.5 µV/LSB)	36 V	140 µs	none
INA228	20-bit (312.5 nV/LSB)	85 V	50 µs	40-bit

Table 2.1.: INA sensor models used in this work [24–26]. The INA219 conversion time is its 12-bit value; the INA226 and INA228 values are their shortest times. Only the INA228 integrates energy on chip; the other two models have no accumulator register.

INA219. The INA219 [24] is the oldest and most widely available module in the family. We started with it because the breakout boards are cheap and easy to find. Its 12-bit ADC provides 10 µV per least significant bit on the shunt channel. The 12-bit conversion time is 532 µs; with shunt and bus converted in turn, this caps the new-conversion rate near 940 Hz, and the measured ceiling is approximately 895 Hz regardless of the bus speed (see the burst capture experiment later in the evaluation). Bus voltage up to 26 V.

The INA226 [25], on the other hand, increases the shunt resolution to 16 bits (2.5 µV/LSB) and reduces the minimum conversion time to 140 µs. It provides a configurable averaging filter (1–1024 samples), which trades temporal resolution for noise reduction. Its bus voltage range is 36 V.

The INA228 [26] is the most recent of the three and also the most interesting one for this work. Its 20-bit shunt ADC resolves 312.5 nV per LSB, and its minimum conversion time is 50 µs. What sets it apart is a 40-bit *energy accumulator* register that integrates power internally at the chip's native conversion rate—this register can be read and reset over I^2C, so it gives an energy measurement that is independent of how often the host reads. Besides, its 85 V bus voltage range gives the largest margin above the 19 V supply of GPU compute modules among the three models.

I^2C interface and timing. All three sensors communicate over the I^2C bus. Each sensor has a configurable 7-bit address, and multiple sensors can share the same bus. A

single read of two 16-bit registers (shunt voltage and bus voltage) takes approximately 1.1 ms at 100 kHz; at 400 kHz, the measured per-sample floor is 0.37–0.42 ms including conversion and software overhead. At low clock speeds, this bus transfer time becomes the bottleneck for the sampling rate, as discussed later in the evaluation. Each sensor also integrates the input voltage over its conversion time before latching the result into a register, so a shorter conversion time means a narrower measurement aperture. In practice the INA228's 50 µs aperture preserves sub-millisecond transients that the INA219's 532 µs aperture averages out; the difference is directly visible when both sensors observe the same current (Subsection 6.5.3).

2.4. Communication Protocols: HTTP and MQTT

The platform uses HTTP for control and MQTT for the continuous data stream. They solve different problems, although both are carried over the same WiFi connection.

2.4.1. HTTP and REST

The Hypertext Transfer Protocol (HTTP) follows a request–response model: a client sends a request (method, path, optional body), the server process it and returns a response (status code, body). Each request is self-contained; the server does not retain the client state between requests.

A Representational State Transfer (REST) interface maps resources to URL paths and uses standard HTTP methods to operate on them: GET retrieves a resource, PUT replaces it, POST creates a new resource or triggers an action, and DELETE removes one. The stateless nature of HTTP makes REST suitable for infrequent, on-demand operations such as reading the current calibration table or changing the sampling rate.

HTTP transport limitation. HTTP/1.1 can maintain persistent connections (keep-alive) to avoid the cost of a new TCP handshake per request. However, each exchange still requires a request header and a response header, which imposes a per-message overhead. On constrained devices, this overhead limits the sustainable message rate. Server-initiated push is possible within the HTTP framework ([3] define a reporting endpoint that triggers periodic push), but the per-message cost remains higher than in dedicated messaging protocols (Section 6.3).

2.4.2. MQTT

MQTT [19] is a lightweight publish–subscribe messaging protocol.[1] Publishers send messages to *topics*; a *broker* receives them and forwards them to interested *subscribers*. In other words, the sensor device does not need informations about the clients which finally store or plot a measurement.

Publish–subscribe model. Eugster et al. [5] identify three dimensions of decoupling in the publish–subscribe model: space (publishers and subscribers do not need to know each other), time (they do not need to be active at the same time), and synchronization (publishing and receiving are asynchronous). For continuous monitoring data, this model has a structural advantage: the device publishes one message per sampling cycle, regardless of how many consumers exist. Adding a new subscriber does not change the device's behavior or cost.

Quality of service and retained messages. MQTT defines three delivery guarantees: QoS 0 (at most once, no acknowledgment), QoS 1 (at least once, acknowledged), and QoS 2 (exactly once, four-way handshake). Higher QoS levels add the protocol overhead. For monitoring data where each message supersedes the previous one, QoS 0 is sufficient—a lost message is replaced by the next one within one sampling period. Meanwhile, a message published with the *retained* flag is stored by the broker and delivered to every new subscriber upon connection, which is useful for configuration data (e.g., the calibration table) that changes infrequently but must be available to any new client.

2.5. Clock Synchronization

When two instruments measure the same physical event independently, their timestamps differ by a *clock offset* and diverge over time by a *clock drift*. If the offset and drift are not corrected, energy attribution to individual events becomes unreliable.

The Network Time Protocol (Network Time Protocol (NTP)) synchronizes a device's clock to a reference server by exchanging timestamped packets and estimating the round-trip delay [17]. The Simple Network Time Protocol (Simple Network Time Protocol (SNTP)) is a simplified subset that uses the same packet format but does not maintain the statistical filter and clock discipline loop of a full NTP client. On a local WiFi network, SNTP typically achieves accuracy on the order of 1 ms to 10 ms, which is sufficient for timestamping monitoring messages but not for aligning sub-millisecond

[1] The project and protocol overview are available at `https://mqtt.org/`.

events across devices. Higher accuracy requires hardware support: the Precision Time Protocol (PTP, IEEE 1588) timestamps packets in the network interface hardware and achieves sub-microsecond accuracy, but it requires PTP-capable network switches; GPS-disciplined clocks achieve similar accuracy without network infrastructure, but require an antenna and a clear sky view. Both add cost.

Post-hoc alignment. An alternative is to skip real-time synchronization entirely and align traces after the experiment. The instruments record data on their own clocks, and an offline step estimates the offset and drift from features that appear in both traces—for example, the load transitions when a workload starts or stops. This approach needs no synchronization infrastructure and no hardware modification. The trade-off is precision: it does leave a residual error, depending on how clear and frequent the shared features are. The method we use is described in the design chapter.

2.6. Inference Workloads

We use machine-learning inference as a representative edge workload. It is often offloaded, its accuracy can be checked, and its energy cost changes strongly between devices. This makes a useful test.

MLPerf Tiny. MLPerf Tiny [1] is a benchmark suite for machine-learning inference on microcontrollers. It defines four tasks (keyword spotting, visual wake words, image classification, anomaly detection), standardized datasets, and reference models. The image classification task uses a ResNet-8 model trained on CIFAR-10 (10 classes, 32×32 pixel images), quantized to 8-bit integers. The reference model has 12 501 632 multiply-accumulate operations per inference. This work uses the ResNet-8 model on the ESP32-S3 microcontroller as one of three evaluation workloads.

Object detection on GPU modules. On the Jetson Orin Nano, the evaluation uses a FasterRCNN-MobileNetV3-Large-FPN object detector from the TorchVision model zoo. A single inference takes approximately 216 ms and consumes 1.14 J at the 19 V power input (Subsection 6.7.1). This workload represents the class of GPU-accelerated inference tasks that dominate edge AI deployments.

Both workloads are measured in a *duty-cycle* configuration: the device alternates between idle and inference at a fixed cadence (e.g., one inference per 500 ms). The idle periods give flat baselines on both sides of each event, so the integration boundary lies in a flat region and the energy result does not depend on its exact position. Back-to-back inference would leave no idle gap, and every boundary would fall on a transition.

3. Related Work

Existing work approaches energy measurement from rather different directions. Some systems build a dedicated instrument, others reuse onboard counters, and measurement testbeds sometimes spend most of their hardware budget only on a common clock. We compare these approaches below. Their assumptions simply do not fit the same edge setup.

This thesis also builds on the offloading work it is meant to serve. SMOOTH [11], budget-adaptive routing [10] and KUT [9] decide at run time where a task should execute, and PTC-IoU [16] routes video frames between a weak and a strong detector without extra training. All of these decisions trade accuracy and latency against energy, and the sections below review how that energy can be measured.

3.1. External Power Measurement Instruments

Table 3.1 compares the external measurement instruments most relevant to this work. The table includes research platforms with open hardware, commercial instruments and one concurrent open-source effort.

EMPIOT. Dezfouli et al. [4] build a single-channel power meter from an INA219 module and a Raspberry Pi. For us, the most relevant part is their *sampling-rate decomposition experiment*: they separate the achievable rate into I^2C bus transfer time and ADC conversion time, and show how the Linux driver stack (BCM vs. i2c-dev) and the bus clock speed affect the result. Their highest rate is approximately 950 sps at 12-bit resolution. We perform a similar decomposition in the evaluation on bare-metal firmware without an operating system driver layer, and extends it with a *new-conversion rate* metric that separates read speed from conversion speed.

RocketLogger. Sigrist et al. [23] design a portable, battery-powered data logger with two current channels (a high-range and a low-range path per channel) that switch automatically with a 1.4 µs transition time. The device achieves 64 kHz per channel and resolves currents at nanoampere scale. Calibration uses a Keithley 2450 source meter as the reference and fits gain and offset—the same two-parameter model used in this

System	Venue	Year	Rate	V_{max}	Key characteristics
EMPIOT	JNCA	2018	0.9 kHz	5.5 V	INA219 + Raspberry Pi; I^2C rate decomposition; single channel
RocketLogger	DATE	2017	64 kHz	5.5 V	Dual-range analog front end; nA-scale resolution; web remote; portable; open hardware
Monsoon HVPM	—	2020	5 kHz	13.5 V	Reference-grade main channel; USB tethered; supplies power to DUT; $1107
Otii Ace Pro	—	—	50 kHz	25 V	Single channel; supplies power; scripting API; $1699/channel
PowerSensor3	ISPASS	2025	20 kHz	—	Analog front end + STM32 ADC; USB host; multi-channel; ∼€100; power error ±1.2–7 W
Shepherd Nova	MobiSys	2025	100 kHz	4.8 V	Energy-harvesting testbed; PTP sync (334 ns); PRU co-processors; open hardware
Bantel et al.	arXiv	2026	1.2 kHz	—	ESP32 + RPi 3; HTTP control; GPIO reset sync; no calibration; no reported accuracy

Table 3.1.: External power measurement instruments (sorted by year).

work (Equation 5.1). The RocketLogger supports web-based remote access and data download. Its voltage range (5.5 V) and single-device form factor distinguish it from the multi-channel, networked architecture of this platform.

Monsoon HVPM. The Monsoon High Voltage Power Monitor [18] is a reference-grade instrument that supplies power to the DUT on one main channel while measuring current at 5 kHz. Its 13.5 V input limit excludes the 19 V barrel-jack supplies of GPU compute modules. The instrument requires a Universal Serial Bus (USB) host connection, which limits its use in automated, multi-device setups. In this work, the HVPM serves as the reference instrument (Section 6.1).

Commercial instruments. The Otii Ace Pro[1] supports up to 25 V and provides a scripting API for automated measurement, but at approximately $1699 per channel multi-node deployments become expensive quickly. Van der Vlugt et al. [29] take a lower-cost approach with PowerSensor3: analog front-end circuits with the

[1]Product information at `https://www.qoitech.com/otii/`.

STM32F411's built-in 10-bit ADC at 20 kHz, connected to a host via USB, for about €100 in parts. Their evaluation reports absolute power errors of ±1.2 to ±7 W, calibrated against a handheld multimeter. Compared to our work, the main differences are the USB-tethered topology and that there is no traceable calibration procedure (no reference-grade instrument, no independent verification round).

Research testbeds. Geissdoerfer et al. [8] extend the Shepherd testbed [7]. The sampling rate stays at 100 kHz per channel; the maximum inter-observer synchronization error drops to 334 ns via PTP. Two real-time co-processors (PRU) handle sampling and digital-to-analog conversion, but the testbed targets batteryless, energy-harvesting IoT nodes and supports supply voltages up to 4.8 V only, which is below the 5 V–19 V range we need. A concurrent effort [2] from Stuttgart measures an ESP32 as the DUT, with a Raspberry Pi 3 as the measurement and control host and a commercial current module. Their synchronization relies on a GPIO reset pin to control the DUT's boot time. The system provides an HTTP interface for remote control but reports neither calibration results nor the measurement accuracy.

No existing instrument combines all of: multi-channel measurement, a 5 V to 19 V input range, a traceable calibration with independent verification, network-accessible control and data, and a parts cost below €50. The gap is most visible at the 19 V supply: the Monsoon HVPM does not reach it, Shepherd Nova and RocketLogger are designed for lower voltages, and PowerSensor3 does not report calibration accuracy at this operating point.

3.2. Software Counters and Onboard Sensors

Many edge devices expose power or energy estimates through software interfaces: NVIDIA's `nvidia-smi` and `tegrastats` read onboard INA sensors on Jetson modules, Intel's Running Average Power Limit (RAPL) interface provides per-domain energy counters on x86 processors, and the Raspberry Pi 5's PMIC reports the power of each regulator output. These interfaces are attractive because they require no external hardware, but several studies have quantified their limitations.

Onboard sensor accuracy. Shalavi et al. [21] measure four generations of Jetson devices and find that onboard sensors under-report the actual input power by up to 50%. After per-device calibration, the residual error is approximately ±3%. Shalavi et al. attribute the gap to losses in the power-supply circuitry and to systematic errors of the built-in sensors.

Sampling coverage. Yang et al. [30] show that `nvidia-smi` on A100 and H100 GPUs updates its power reading only during 25% of the runtime. A single naïve energy measurement based on one reading can err by up to 70%. They propose oversampling and statistical correction, but the underlying limitation—that the onboard sensor's update rate is fixed by the hardware and cannot be controlled by the user—remains.

In summary, the above findings mean that onboard sensors cannot serve as the sole energy reference for offloading decisions. An external measurement at supply input is needed at least once per device to establish a calibration baseline. Our platform provides this capability. What is more, it can run concurrently with onboard recording, so that both views of the same workload are available for comparison.

3.3. Time Synchronization in Measurement Systems

Attributing energy to a specific task requires knowing *when* the task started and ended on the power trace. The literature uses hardware synchronization, injected markers, GPIO timestamps, and post-hoc signal alignment. These categories overlap somewhat, especially when a system timestamps a marker in hardware and still aligns traces afterwards.

Hardware synchronization. FlockLab 2 [27] provides sub-microsecond timing using satellite-disciplined clocks on each observer node (about 50 ns for GNSS-disciplined observers). Each observer records power at 64 kHz locally; data is downloaded to the server after the experiment. Shepherd [7] targets 1 µs inter-observer synchronization using PTP with hardware timestamping (maximum observed error 2.4 µs), combined with a custom kernel module that couples the Linux clock to the PRU co-processor every 100 ms; Shepherd Nova [8] reduces the maximum synchronization error to 334 ns. Both systems require infrastructure (satellite antennas or PTP-capable switches) that adds cost and constrains deployment locations. Geng et al. [12] show that Huygens achieves clock synchronization to within 100 ns in data centers, using less than 0.44% of each server's CPU time.

Injected markers. When no hardware synchronization is available, some systems inject artificial power signatures into the supply line and use cross-correlation to align the power trace to a host-side event log. The BattOr system [20] modulates the phone's camera-flash LED (about 2 W) with a pseudorandom sequence and cross-correlates the resulting power signature with the LED driver calls that the phone's kernel tracer (ftrace) timestamps. No hardware modification is needed, but the method requires a controllable load and kernel-level tracing on the measured device itself.

GPIO timestamps. MLPerf Power [28] requires the DUT firmware to toggle a GPIO pin at the start and end of each inference. The energy monitor records these pin transitions alongside its power samples; an I/O manager isolates the DUT from the host. This approach is precise but invasive: the DUT's firmware must be modified to produce the synchronization signal, and the I/O manager hardware must be physically connected.

Post-hoc signal alignment. Quanto [6] takes a different approach: it attributes energy to software activities (interrupt handlers, tasks, network packets) by tracking which activity was running when the power measurement was taken, using a per-activity energy counter (iCount). This avoids explicit time synchronization but requires deep integration with the operating system.

For this work we selected a post-hoc approach that requires no hardware modifications to the DUT, no synchronization infrastructure, and no firmware changes on the measured device. The key observation is that workload transitions produce natural current transitions that are visible on the power trace. These transitions serve as alignment markers: cross-correlation of the load pattern between the host-side event log and the power trace yields the clock offset and drift. Of course, the trade-off is precision: the residual alignment error (median 132 µs, measured in the evaluation chapter) is three orders of magnitude larger than FlockLab's satellite-disciplined result, but it is achieved at zero infrastructure cost and with zero DUT modification. For the workloads in this thesis (inference events of approximately 0.2 s to 1.4 s), this residual is small relative to the event duration.

4. System Design

The design started from a practical question: where can the power meter be inserted without requiring a change inside every measured device? The supply cable was the only boundary common to the ESP32-S3, Pi 5 and Jetson. From this choice follow the voltage range, the channel count and also most of the communication design discussed below. Clock alignment was added later because a correct energy integral is not yet a task-energy value when the two clocks disagree.

4.1. Requirements

An offloading decision considers task accuracy, latency and input energy. Software can provide accuracy and latency. For input energy, many edge devices report no power data at all, and those that do report only a fraction of the input power, as discussed in Section 3.2. An external measurement platform must fill this gap.

The following requirements came from the offloading use case and from the limitations found in existing instruments.

R1 **Measure at the device power input.** The platform must measure the total power that enters the device through its supply cable. Onboard sensors under-report the input power by up to 50% on Jetson devices [21].

R2 **Wide cover range** Common edge devices operate at 5 V (microcontrollers, single-board computers) or 19 V (GPU modules with barrel-jack supplies). The sensor inputs must accept this range.

R3 **Multiple channel supports and channel modularization** An offloading comparison needs at least two candidate devices plus one baseline. All channels must share a common time reference so that their measurements are comparable.

R4 **Adjustable time resolution.** Continuous monitoring at 1–50 Hz shows the live power curve. Kilohertz-rate capture records transient events such as a single inference (typically 50–500 ms), which are invisible at low rates.

R5 **Allow remote access and control.** Experiment scripts must be able to read measurements, change settings, and start captures over the network without physical access to the device.

R6 **Budget limitation** Multi-node deployments become practical only if each measurement node is inexpensive. The target is below €50 per node.

Together these requirements rule out several existing choices. Consumer USB power meters have a single channel and no programmatic interface (violates R3 and R5). Tethered laboratory instruments such as the Monsoon HVPM supply and measure only one main channel and require a USB host connection (violates R3 and R5) [18]. Software-only methods based on onboard sensors cover only part of the input power and their values change with the load (violates R1) [21].

4.2. System Architecture

For implementation, we seperate the platform roughly into device, broker, and application layers (Figure 4.1).

The **device layer** contains the power meter and the DUTs. The power meter is an ESP32-C6 microcontroller that reads three shunt sensors over I^2C and publishes the measurements over WiFi. The three DUTs in this work are an ESP32-S3 microcontroller (5 V), a Raspberry Pi 5 single-board computer (5 V), and a Jetson Orin Nano edge graphics processing unit (GPU) (19 V).

The **broker layer** is an MQTT broker (Mosquitto)[1] that runs on a host PC. The broker receives all measurement data from the power meter and distributes it to any number of subscribers. Moreover, it forwards commands from applications to the device.

At the **application layer**, a browser dashboard gives live monitoring, the recording server writes SQLite, and Python scripts run the automated experiments. They receive the same stream from the broker, but do it independently.

This separation follows the Tenet principle [13]: the device performs only local acquisition; cross-device processing belongs to the higher layers. Because the broker handles distribution, the power meter does not need to know how many applications consume its data. Adding or removing a subscriber does not change the device's workload.

[1] Mosquitto is the Eclipse Foundation's open-source MQTT broker: `https://mosquitto.org/`.

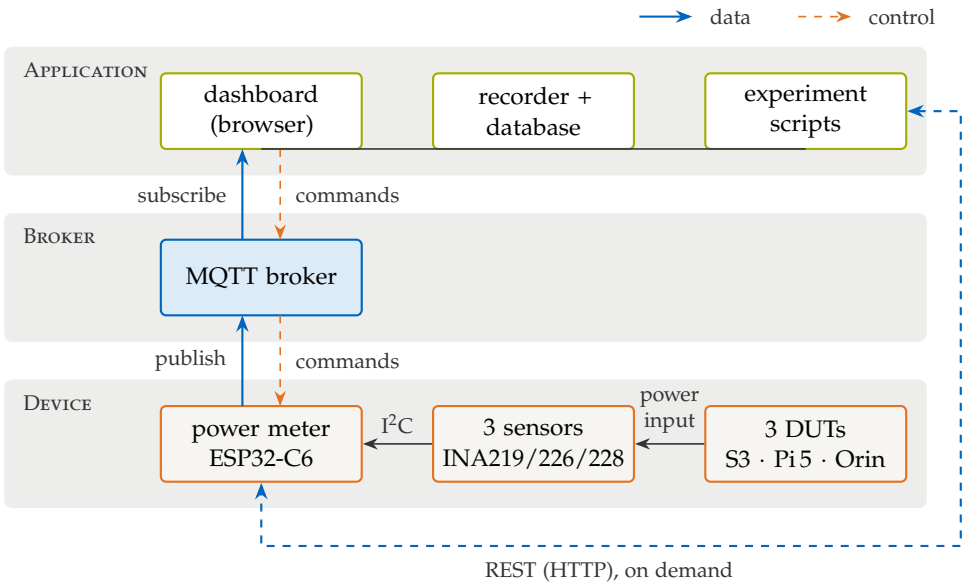


Figure 4.1.: Three-layer architecture. Solid blue arrows carry measurement data (MQTT publish/subscribe); dashed arrows carry control commands. The power meter sends its data only to the broker, and the broker sends the data to all applications. The request-response path (REST) bypasses the broker.

4.3. Measurement Hardware

Each DUT has one dedicated measurement channel. A channel consists of a shunt resistor in series with the DUT's power input and a current-sense amplifier that reads the voltage drop across the shunt (Figure 4.2).

Figure 4.3 shows how one such channel is wired in practice, with the ESP32-S3 supply as the example. The DUT's supply cable is opened at a USB-A adapter, so no modification inside the DUT is necessary; the same insertion works with a USB-C adapter cable (Pi 5) and a barrel-jack adapter cable (Jetson).

On the assembled bench, the ampere-level power path never crosses the breadboard: it enters each sensor module through screw terminals and short, thick wires. The breadboard carries only the I^2C side. A photograph of the assembled bench with all three channels connected is in Appendix C.

The sensor type on each channel matches the current range of the DUT. Table 4.1 lists the three channels.

The INA219 has the largest shunt (100 mΩ) and therefore the highest sensitivity. It resolves currents above approximately 1 mA, which is sufficient for the ESP32-S3

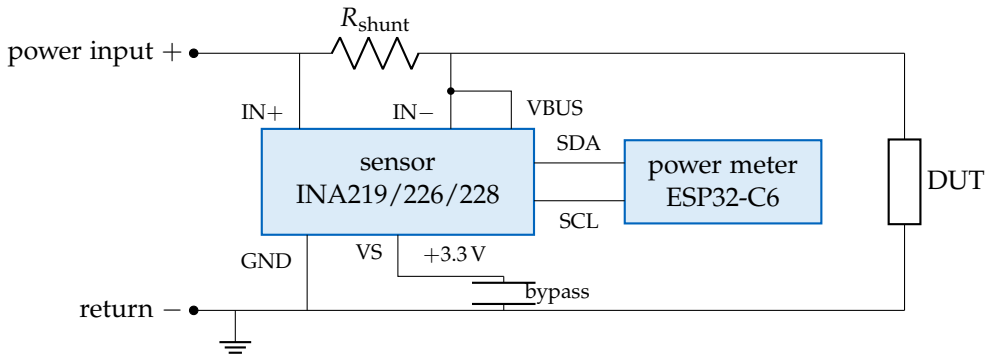


Figure 4.2.: One measurement channel. The shunt resistor sits in series with the DUT's power input; the sensor reads the shunt voltage through its Kelvin taps and the bus voltage on the load side, and it reports both to the ESP32-C6 over I^2C.

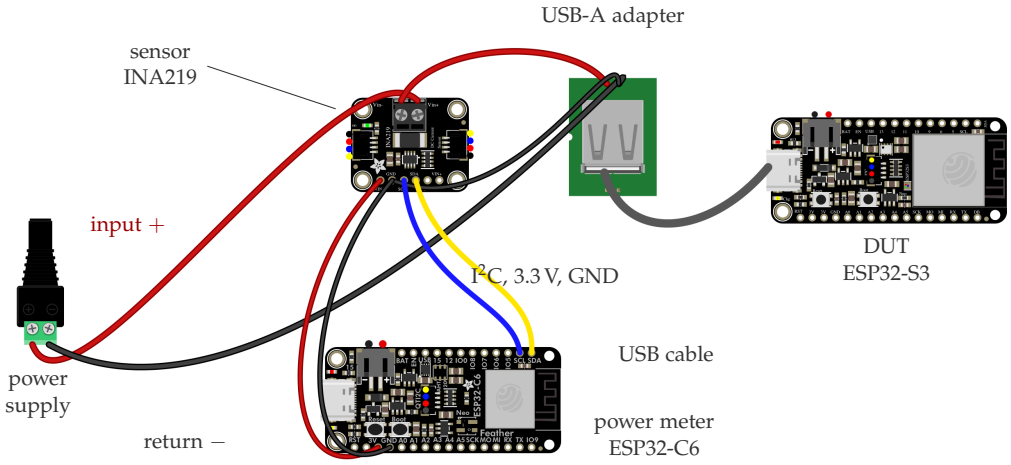


Figure 4.3.: Physical wiring of one channel: the INA219 sensor on the ESP32-S3 power input. The supply cable is opened at a USB-A adapter, the current path enters the sensor module through its screw terminals, and the I^2C side connects the sensor to the power meter.

Power input	Sensor	ADC	Shunt	Current range	Voltage
ESP32-S3, 5 V	INA219	12-bit	100 mΩ	1 A	5
Raspberry Pi 5, 5 V	INA226	16-bit	2 mΩ	20 A	5
Jetson Orin Nano, 19 V	INA228	20-bit	15 mΩ	2.7 A	19

Table 4.1.: Three input channels. The shunt resistance determines the current range; the sensor resolution determines the smallest resolved current step.

(120–250 mA). The INA226 uses a 2 mΩ shunt to reach a 20 A range, which covers the Raspberry Pi 5 (0.5–2.1 A under full central processing unit (CPU) load). The trade-off is a higher noise floor: the smallest resolved current on this channel is approximately 5 mA. The INA228 uses a 15 mΩ shunt, has 20-bit resolution, and includes an on-chip energy accumulator that integrates current independently of the sampling rate [26]. It covers the Jetson Orin Nano (0.3–1.2 A at 19 V).

All three sensors connect to the ESP32-C6 over a single shared I^2C bus. The complete power meter—one ESP32-C6 board and three sensor modules—costs approximately €27 in retail parts. Table 4.2 summarizes the parts cost.

Part	Qty	Price
ESP32-C6 board with display	1	∼€12
INA219 sensor module	1	€1.50
INA226 sensor module	1	€3.00
INA228 sensor module	1	€5.50
Adapter boards and cables	6	∼€5.00
Total		**∼€27**

Table 4.2.: Parts cost for a three-channel power meter. Retail prices from August 2026. The reference instrument that is necessary for calibration is not included.

4.4. Communication Design

The communication is divided by purpose, which keeps the two problems separate. Request-response over HTTP is used for an occasional operation, while publish-subscribe over MQTT carries the continuing samples. Figure 4.4 shows the two patterns.

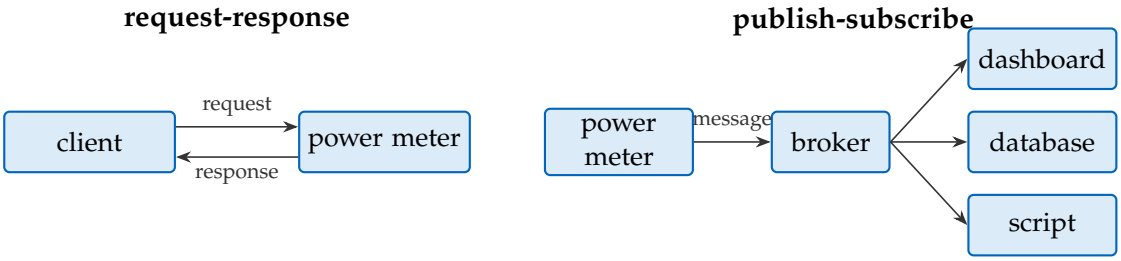


Figure 4.4.: The two communication patterns. The REST interface carries one value or one control command per request; the MQTT interface carries the continuous data stream from the power meter to many consumers.

4.4.1. REST Interface

The power meter runs an HTTP server with a versioned REST interface (prefix `/api/v1/`). REST handles two tasks: control commands and on-demand data queries.

A control command changes the device state. For example, an experiment script sets the sampling rate to 50 Hz by sending `PUT /api/v1/rate` with `{"hz":50}`. The response confirms whether the device accepted the change. This confirmation is necessary: the script must verify that the new rate is active before it starts a measurement.

An on-demand query reads a single value. For example, `GET /api/v1/rate` returns the current sampling rate, the per-channel ADC integration time, and the integration duty cycle. These queries are infrequent and do not affect the measurement loop.

In the current implementation the device exposes 9 endpoints. The recording server on the host PC adds 20 endpoints for experiment control, DUT management, and history queries. Appendix A lists all endpoints with their methods, parameters, and response formats.

4.4.2. MQTT Interface

MQTT handles the continuous data stream. The power meter publishes each measurement as a message to the broker. Any number of subscribers receive it. This arrangement has three properties that Eugster et al. identify as the defining characteristics of publish-subscribe systems [5]:

- *Space decoupling*: the publisher does not know the subscribers.
- *Time decoupling*: the broker retains the last message on each topic, so a subscriber that connects later still receives the most recent value.
- *Synchronization decoupling*: publishing does not block the device. The power meter continues sampling immediately after it hands the message to its local MQTT

client.

Topic scheme. MQTT topics follow a two-level hierarchy:

`{device_type}/{device_id}/{stream}`

Table 4.3 lists the device-side topics; the full topic tree appears in Appendix B. The stream name indicates the content: `telemetry` carries the continuous monitoring messages, `cal` carries the retained calibration table, `cmd` receives commands, `burst` and `trigger` carry binary capture data, and `acc` carries accumulator readings.

Topic	Dir	Content
`powermeter/{id}/telemetry`	▷	monitoring messages (JSON, continuous)
`powermeter/{id}/cal`	▷	calibration table (JSON, retained)
`powermeter/{id}/cmd`	◁	commands from host (JSON)
`powermeter/{id}/burst`	▷	streaming capture batches (binary)
`powermeter/{id}/trigger`	▷	transition capture data (binary + JSON)
`powermeter/{id}/acc`	▷	energy accumulator readings (JSON)
`workload/{id}/cmd`	◁	workload commands for a DUT
`workload/{id}/state`	▷	current workload state (retained)

Table 4.3.: MQTT topic scheme. Topics with ▷ are published by the device; topics with ◁ are subscribed by the device.

4.4.3. Message Format

Each monitoring message is a JavaScript Object Notation (JSON) object with 13 top-level fields and one object per channel. Table B.2 and Table B.3 in Appendix B list every field with its type, together with a recorded example message.

Two timestamps appear in each message. The field `mono_us` is the ESP32-C6's monotonic clock. It increases at a constant rate and never jumps, so it is suitable

for measuring intervals between samples. The field `epoch_us` is the wall-clock time obtained from SNTP. It is necessary for absolute alignment with DUT logs (Section 4.6). Both timestamps are recorded together when the message is built, after the sensor readout and before any network operation.

Transition capture and streaming capture use a binary format for efficiency. Each binary batch starts with a 16-byte header (two magic bytes, device address, flags, sequence number, sample count, start timestamp, bus voltage) followed by an array of (`dt_us`, `current_mA`) pairs, each 8 bytes. A JSON summary at the end of each capture reports the total sample count and the trigger parameters. Appendix B documents every field of every message type.

4.4.4. Protocol Selection

The protocol choice is based on implementation and measurement, following the principle of Hui and Culler [14]: evaluate protocols by building complete implementations and testing them, not by comparing feature lists.

We ran a transport benchmark on the power meter itself—the same ESP32-C6 board, the same JSON payload, 48 runs of 20 s each. The benchmark measures the median time the device spends handling one message, excluding sensor acquisition. Table 4.4 shows the results.

Method	Handling time (ms)	Max rate (Hz)
MQTT PUBLISH	0.7	50.0
WebSocket	1.2	50.0
HTTP GET (polling)	4.5	43.0
HTTP POST, keep-alive	22	34–37
HTTP POST, new conn.	84	11.5
No transport (baseline)	0.001	—

Table 4.4.: Transport benchmark results. All measurements on the same ESP32-C6 board with the same payload. "Handling time" is the median time the device is occupied per message; "max rate" is the delivered rate at the 50 Hz setting in this benchmark, an upper bound that holds under favorable network conditions (Chapter 6). Serialization itself takes about 1.2 ms and is not part of the handling time.

An MQTT PUBLISH occupies the device for 0.7 ms per message. At 50 Hz (the maximum configured rate), one message is due every 20 ms. MQTT therefore uses 3.5% of the sampling period for transmission. An HTTP POST with a persistent TCP

connection takes 22 ms—longer than the 20 ms period itself. Continuous monitoring at 50 Hz over HTTP is therefore not feasible on this device.

The difference arises because each HTTP request waits for the application-layer response before the device can proceed. An MQTT PUBLISH at QoS 0 does not wait for a response; the device hands the message to its local client library and returns immediately.

WebSocket and raw UDP also sustain 50 Hz (Chapter 6), so the decision against them is not about speed. Both are point-to-point: the device would maintain one connection or destination per consumer, and every added consumer would add device work. UDP additionally gives no delivery guarantee, so a lost message disappears silently. MQTT moves the fan-out to the broker and retains the calibration table for late subscribers; the device workload stays independent of the number of consumers.

The scalability results from Chapter 6 confirm the design: eight MQTT subscribers all receive 10.0 Hz with no increase in the device's publish time. Eight simultaneous REST clients cause a 95th-percentile response time of 1084 ms and 5.6% of requests fail.

4.5. Acquisition Modes

The device supports four mutually exclusive acquisition modes: averaged monitoring, burst capture, streaming capture, and transition capture. The operator selects one at a time through a command on the MQTT control topic. Table 4.5 compares averaged monitoring and transition capture.

	Averaged monitoring	Transition capture
Operation	continuous	on demand
Rate	1–50 Hz	2 kHz practical (2.4 kHz absolute)
Granularity	coarse (long average)	fine (individual samples)
Output	live power curve	one event window
Trigger	none (periodic)	current threshold + sustain

Table 4.5.: Comparison of averaged monitoring and transition capture.

4.5.1. Averaged Monitoring

Averaged monitoring is the continuous mode. The power meter samples each channel at a configured rate between 1 Hz and 50 Hz and publishes one monitoring message per sample. Each message contains the mean current over the ADC's integration time.

The firmware adjusts the ADC configuration to match the sampling rate. Each sensor type has a table of (averaging count, conversion time) pairs. When the operator sets a new rate, the firmware selects the longest integration time that fits within 80% of the sampling period. This adjustment maximizes noise rejection at each rate.

As a concrete example, at 10 Hz the INA226 uses 64-fold averaging with a 588 µs conversion time. The total integration time is $64 \times (588 + 588)$ µs $= 75.3$ ms out of a 100 ms period, which gives a 75% integration duty cycle. At 50 Hz the averaging must be reduced to fit within the 20 ms period, and the integration duty cycle decreases.

Averaged monitoring serves two purposes: live observation through the dashboard and persistent recording through the database. Both receive the same stream from the broker.

4.5.2. Transition Capture

Transition capture is an on-demand, high-rate mode. It records one event window at a practical 2 kHz (absolute limit: 2.4 kHz) when a current transition occurs.

A current transition is a step change in the DUT's current consumption. Figure 4.5 shows why transition capture is necessary. At 10 Hz, a transition that lasts 10 ms has a negligible effect on the averaged value and does not appear in the monitoring stream. To record the transition itself, the power meter must sample at a kilohertz rate.

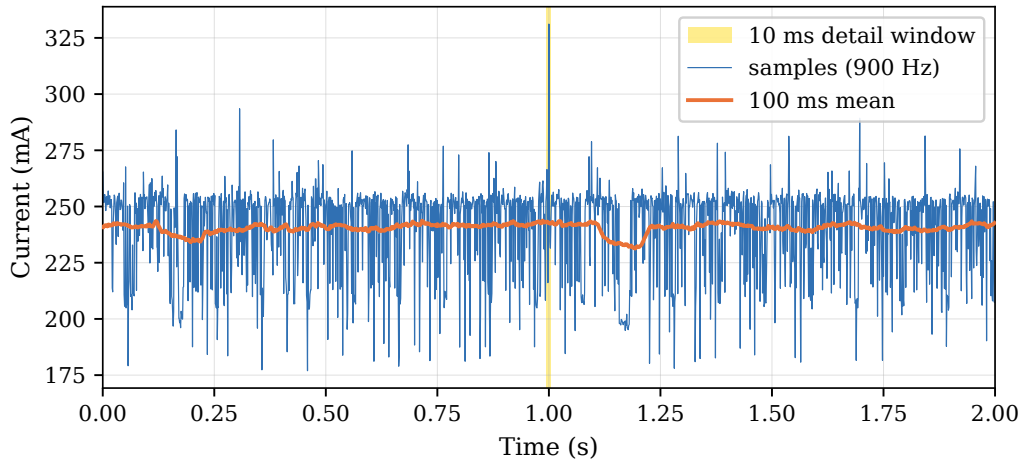


Figure 4.5.: ESP32-S3 input current during WiFi transmission (INA219 channel, 900 Hz; 2 s excerpt of a 15 s capture). The capture shows individual current transitions up to 331 mA; the 100 ms window mean—what 10 Hz monitoring reports—stays near 240 mA and shows none of them. The highlighted band marks a 10 ms detail window around the largest transition.

The capture sequence is the following:

1. The host sends a trigger command with five parameters: threshold current, sustain window, pre-trigger window, post-trigger window, and sampling rate.
2. The power meter enters a fast acquisition loop and fills a ring buffer (the pre-trigger buffer).
3. When the moving mean over the sustain window exceeds the threshold, the trigger fires.
4. The power meter continues sampling for the post-trigger duration.
5. The device publishes the complete capture (pre-trigger and post-trigger data) as binary batches, followed by a JSON summary.

Figure 4.6 shows the trigger concept.

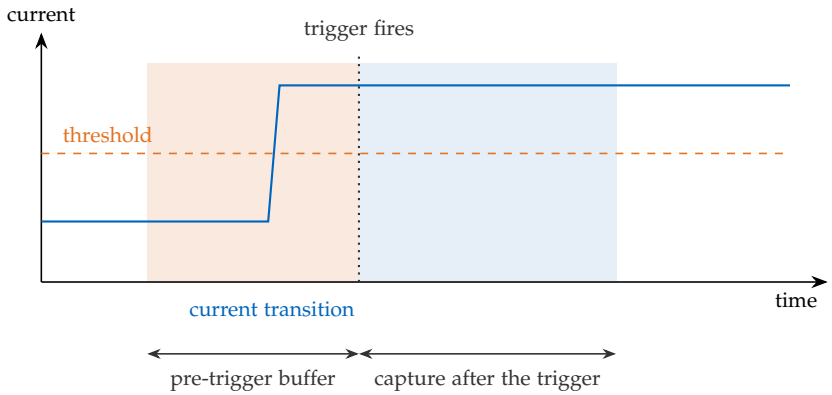


Figure 4.6.: Transition capture. The trigger fires when the moving mean stays above the threshold for the sustain window; this confirmation is the gap between the current transition and the trigger instant. The pre-trigger buffer (shaded orange) preserves the baseline from before the transition.

The pre-trigger buffer turned out to be the key element of this design. It preserves the baseline current from before the transition. Without it, the trigger fires only after the transition has been sustained, and the onset of the event is missing from the capture. Because the buffer is a ring buffer, its oldest samples are overwritten continuously. This causes no loss as long as the pre-trigger window is longer than the sustain window: the trigger fires at most one sustain window after the transition starts, so the onset is still in the buffer at the trigger instant.

The trigger criterion—the moving mean stays above the threshold over the whole sustain window—rejects short noise bursts. The idle ESP32-S3, for instance, produces WiFi bursts that reach 218 mA, but its moving mean over a 150 ms window never exceeded 170.3 mA in a 15-minute idle scan. With a 200 mA threshold and this sustain

window, the power meter recorded zero false triggers in that scan (Chapter 6).

4.6. Time Synchronization

Energy attribution assigns a portion of the measured energy to a specific task. This requires two pieces of information: the power trace from the power meter and the task schedule from the DUT's log. These come from two independent clocks. If the clocks are not aligned, the integration window shifts relative to the true task boundaries, and the energy value is incorrect.

The idea of our method is simple: the DUT runs a short, known load pattern, this pattern is visible in the power trace, and matching the two tells how far the two clocks are apart. No extra wire and no change inside the DUT are needed.

We first formulate this clock problem, then explain the load markers and finally how an unreliable alignment is rejected.

4.6.1. Three Independent Clocks

In practice each measurement contains independent timestamps from the following clocks:

1. The ESP32-C6 clock on the power meter, which timestamps each sample (`mono_us` and `epoch_us`).
2. The DUT's clock, which timestamps each task start and end in the DUT's log.
3. The host PC's clock, which timestamps each message at arrival in the recording server.

Each pair of clocks has an unknown offset θ (a constant shift) and an unknown drift rate ρ (a slow change of this shift). The effect on the energy value is easy to state. Consider a task with active power P_a, idle power P_i, and duration T. If the integration window is shifted by δ, the integral swaps δ of task time for δ of idle time, so the absolute energy error is $\delta\,(P_a - P_i)$. Relative to the task energy $P_a T$, this gives:

$$\varepsilon = \frac{P_a - P_i}{P_a} \cdot \frac{\delta}{T} \tag{4.1}$$

Table 4.6 shows the maximum alignment error δ for several combinations of task duration and acceptable energy error. On the Jetson Orin Nano ($P_a \approx 15\,\mathrm{W}$, $P_i \approx 5\,\mathrm{W}$), a 200 ms inference task with a 2% energy error budget requires $\delta \leq 6\,\mathrm{ms}$.

The common alignment methods did not fit our setup for the following reasons:

- The platform must not modify the DUT's hardware. Trigger lines and shared general-purpose input/output (GPIO) pins require internal wiring. Future DUTs

Task duration T	$\varepsilon = 1\%$	$\varepsilon = 2\%$	$\varepsilon = 5\%$
500 ms	7.5	15.0	37.5
200 ms	3.0	6.0	15.0
50 ms	0.75	1.5	3.75

Table 4.6.: Maximum alignment error δ (in ms) for a Jetson Orin Nano task ($P_a = 15\,\text{W}$, $P_i = 5\,\text{W}$), computed with Equation 4.1.

may include various types of device like network switches and routers where such wiring is inconvenient.

- The ESP32-C6's SNTP implementation timestamps packets at the protocol stack level. The measured jitter is on the order of 10 ms. This does not meet the 6 ms budget for a 200 ms task at 2% error.

Clock synchronization alone (NTP or SNTP) reduces the offset but does not eliminate it. The residual jitter exceeds the error budget for short tasks. Sub-microsecond synchronization (Huygens [12], Sundial [15]) is designed for data-center network infrastructure, and still depends on the quality of the DUT's time client.

4.6.2. Load Marker Method

We use load markers to align the clocks after each experiment run. It is a post-processing step, so a failed marker does not stop the measurement but it can make that run unusable.

At the beginning and the end of each run, the DUT executes a known load pattern: three full-load pulses with durations of 100, 200, and 100 ms, separated by 100 ms gaps. The DUT records the start timestamp of each pulse in its own clock. The analysis pipeline then locates the same pattern in the power trace: the expected pattern is slid along the trace, and the position of the best match gives the offset θ between the two clocks. This sliding comparison is a standard cross-correlation.

Two markers (one at the start, one at the end of the run) also determine the drift rate ρ:

$$\rho = \frac{\theta_{\text{end}} - \theta_{\text{start}}}{t_{\text{end}} - t_{\text{start}}} \tag{4.2}$$

The pattern is chosen so that normal workloads are unlikely to produce the same shape by chance: three pulses with two distinct durations at a fixed spacing give the cross-correlation one dominant peak.

Each type of DUT produces the marker pattern in a different way, but in all cases, no hardware modification is necessary:

- **Jetson and Raspberry Pi**: a CPU stress pulse (the `stress` command or a tight arithmetic loop). The DUT records timestamps with `clock_gettime(CLOCK_MONOTONIC)`, which has nanosecond resolution.
- **Router** (if login is available): an SSH-triggered CPU pulse, recorded by the triggering script.
- **Network switch** (no login): two external hosts send traffic bursts through the switch; the sending host records the timestamps.

Two-level cross-correlation. The analysis pipeline performs the cross-correlation at two levels of precision:

Level 1 (coarse) operates on the 50 Hz monitoring stream. The template is the expected power shape of the three-pulse pattern, sampled at 50 Hz. The precision of a discrete cross-correlation is limited to $\pm$ half the sampling period, which is $\pm$ 10 ms at 50 Hz. Interpolation on the correlation peak can improve this, but the result remains in the millisecond range.

Level 2 (fine) operates on transition capture data. The power meter captures the marker pulses at 2 kHz in practice (absolute limit: 2.4 kHz). At 2 kHz, the discrete precision limit is $\pm$ 0.25 ms. The measured 95th-percentile alignment error at this rate is 0.47–1.07 ms (Chapter 6).

The coarse level is sufficient for tasks longer than approximately 500 ms. The fine level is necessary for tasks in the 50–200 ms range, such as a single inference on the Jetson Orin Nano (median duration 216 ms).

4.6.3. Qualification Criteria

Not every alignment is reliable. The analysis pipeline computes a peak ratio and a start–end consistency value for each run, then accepts or rejects the alignment.

The first metric is the **peak-to-second-peak ratio** of the cross-correlation. A high ratio means that the marker pattern was found at exactly one location in the power trace. A low ratio means that the pattern is ambiguous—for example, if the DUT's normal workload happens to produce a similar shape.

The second metric is the **consistency** between the start marker and the end marker. After correcting for drift with Equation 4.2, the two estimates of θ should agree within the alignment precision. A large difference indicates that one of the markers was not correctly identified.

If either metric falls below its threshold, the pipeline marks the run as *unqualified* and excludes it from the analysis. This gives each experiment run an explicit quality label. The operator does not need to inspect alignment plots manually to decide whether a run is usable.

The next chapter goes into the implementation, where the single-core firmware has to sample, serve network requests and refresh its display at the same time.

5. Implementation

The implementation did not grow in the same order as the architecture diagram. We first needed stable sensor readout, then a way to see the values, and only afterwards the experiment recorder and alignment scripts became useful. The presentation below follows approximately this path from the ESP32-C6 firmware to the server and offline analysis. It also records a few details—display stalls and WiFi bursts in particular—which looked small during coding but were visible in the measurements.

5.1. Power Meter Firmware

The firmware runs on ESP32-C6 microcontroller (RISC-V, single-core, 160 MHz) and uses Arduino framework to realize the device functions. PubSubClient handles MQTT, ArduinoJson is used for serialization, and each INA model has a small driver. The firmware connects to the MQTT broker over WiFi; broker discovery uses mDNS (hostname `thesis-hub.local`) with a static IP fallback.

5.1.1. Channel Management

The channel manager scans I^2C bus addresses `0x40` through `0x4F` at startup and every 3 s during operation. Each detected sensor is matched against the board defaults in Table 5.1, which supply the shunt resistance, full-scale current, ADC range bit, and the minimum active current for each sensor model.

Model	R_{shunt} (Ω)	I_{max} (A)	ADC range	I_{min} (mA)
INA219	0.100	1.00	n/a	10.0
INA226	0.002	20.48	n/a	10.0
INA228	0.015	2.73	1	10.0

Table 5.1.: Board defaults per sensor model.

I_{min} defines the current below which the firmware reports a channel as inactive. The server uses this flag to distinguish a powered-off DUT from one at idle.

5.1.2. Rate-Adaptive ADC Configuration

The ADC conversion parameters—number of averages and conversion time per sample—follow the configured sampling rate. Each sensor driver carries a lookup table of parameter combinations. The firmware selects the entry with the longest integration time that completes within 80% of the sampling period, so the sensor averages over as much of each cycle as the budget allows. Table 5.2 shows the resulting configurations at four representative rates.

Rate	INA226	INA228	INA219
2 Hz	512 × 332 µs (340 ms)	256 × 540 µs (276 ms)	128 avg (136 ms)
10 Hz	64 × 588 µs (75 ms)	64 × 540 µs (69 ms)	64 avg (68 ms)
20 Hz	64 × 204 µs (26 ms)	64 × 280 µs (36 ms)	32 avg (34 ms)
50 Hz	16 × 332 µs (10.6 ms)	64 × 84 µs (10.8 ms)	8 avg (8.5 ms)

Table 5.2.: ADC configuration at four sampling rates. Each cell shows averages × conversion time (total integration time).

The selection runs on sensor attach and on every rate change. After a high-rate capture, the firmware restores the monitoring configuration.

5.1.3. Per-Channel Calibration

Each channel carries two calibration parameters: a gain factor (default 1.0) and a current offset (default 0 mA). We apply the inverse of the error model in the firmware:

$$I_{\text{corrected}} = \frac{I_{\text{raw}} - \mathit{offset}}{\mathit{gain}} \tag{5.1}$$

Both the published current value and the energy integrator use the corrected result.

The parameters are stored in non-volatile storage (NVS) with keys derived from the I^2C address (`g44` for the gain at address `0x44`, `o44` for the offset). We chose to address by bus address rather than by channel index, because a bus rescan that changes the detection order would otherwise invalidate the stored values—this actually happened once during development when we added a fourth sensor to the bus.

Calibration can be modified through MQTT JSON commands: `set_cal` writes gain and/or offset for one address, `clear_cal` resets one channel to identity, and `get_cal` returns the full table. After any change, the firmware publishes the complete calibration table as a retained MQTT message so that a newly connected client receives the current state at once.

5.1.4. Sampling Cycle

Each cycle reads all attached sensors, serializes the result as a JSON monitoring message, and publishes it over MQTT. The firmware also reports its acquisition, publication, network-service and display timings (Table 5.3), which serve as the timing budget for the evaluation in Chapter 6.

Field	Unit	Description
`acq_us`	µs	Time to read all sensors over I^2C
`pub_us`	µs	Time to serialize and publish the previous message
`net_us`	µs	Network servicing between cycles (MQTT keepalive, REST, OTA)
`ui_us`	µs	LCD refresh cost

Table 5.3.: Timing fields reported in each monitoring message.

The LCD refresh runs on a separate 500 ms timer but shares the single CPU core with the sampling loop. One refresh costs approximately 70 ms, so a refresh that starts near a sampling deadline delays that cycle; the rate experiments in the evaluation therefore run with the display off (Subsection 6.4.1).

5.1.5. High-Rate Capture

For rates above averaged monitoring, the firmware implements three mutually exclusive modes: burst and streaming capture at configured rates up to 3000 Hz, and transition capture at a practical 2 kHz (absolute limit: 2.4 kHz), beyond the 50 Hz ceiling of averaged monitoring (Table 4.5).

Burst capture. Reads a single channel in the sensor's fastest ADC mode into a RAM buffer of at most 100 kB (approximately 8500 samples at 12 B each). The normal sampling cycle pauses during the capture. The result is published as JSON chunks on the `powermeter/<id>/burst` topic. MQTT commands received during a capture are deferred, because an NVS write would stall the single-core microcontroller for up to tens of milliseconds.

Streaming capture. Reads a single channel at 100 Hz to 3000 Hz for up to 120 s. Samples leave the device in binary batches during acquisition rather than being buffered in RAM, which removes the memory constraint of burst capture. Each batch holds up to 480 samples ($480 \times 8 + 16 = 3856$ B, within the 8 kB MQTT transmit buffer). The batches are published on the same burst topic.

Transition capture. As a distinct mode, transition capture reuses the streaming engine with a trigger mechanism (Subsection 4.5.2). The firmware acquires samples into a circular buffer that retains the most recent `pre_ms` milliseconds. The trigger fires when the moving mean current over a `sustain_ms` window exceeds a configurable threshold. On trigger, the firmware publishes the pre-trigger and post-trigger window on the `powermeter/<id>/trigger` topic, followed by a JSON summary that reports the trigger instant and the largest moving mean observed while armed.

Streaming capture and transition capture use the binary batch format in Table 5.4.

5.1.6. Binary Batch Format

Streaming capture and transition capture share the binary batch format shown in Table 5.4. Bus voltage appears once in the header because it remains stable within a single batch window (tens to hundreds of milliseconds). All multi-byte fields use little-endian byte order.

Offset	Size	Type	Field
Header (16 B)			
0	2	char[2]	Magic (`'S','1'`)
2	1	uint8	I^2C address
3	1	uint8	Flags
4	2	uint16	Sequence number
6	2	uint16	Sample count n
8	4	uint32	Start time t_0 (µs)
12	4	float32	Bus voltage (V)
Per sample ($\times$ n, 8 B each)			
0	4	uint32	Δt from t_0 (µs)
4	4	float32	Current (mA)

Table 5.4.: Binary batch format. Header: 16 B; each sample: 8 B.

5.1.7. Clock Pairing

Each monitoring message carries two timestamps taken back to back: `epoch_us` from the SNTP-disciplined wall clock (`gettimeofday`), and `mono_us` from the monotonic microsecond counter (`micros`). Sample offsets in high-rate captures are expressed as deltas from `mono_us`. A single pair per message maps any sample offset to an absolute time without assumptions about network delay.

5.1.8. Energy Accumulator Readout

The INA228 integrates power in hardware (register `ENERGY`, 40-bit) at the conversion rate, independent of how often the firmware reads the sensor. Two MQTT commands expose this register:

- `{"acc":{"addr":68}}` reads the accumulated energy.
- `{"acc":{"addr":68,"reset":1}}` resets the accumulator (`CONFIG.RSTACC`), then reads.

The reply carries both the hardware value and the firmware's own trapezoidal integral over the same interval. The two integrals are computed from the same analog front end but on different time bases—the hardware one at the conversion rate, the firmware one at the sampling rate—so their agreement bounds the error that the sampled integral commits.

5.2. Workload Firmware and Marker Agent

The DUT workload firmware runs on the ESP32-S3 and receives commands over MQTT. Mark, flow and task modes were added for different stages of the experiments.

Mark mode. Generates the three-pulse load marker pattern (100 ms, 200 ms, 100 ms bursts separated by 100 ms pauses) described in Subsection 4.6.2. The marker uses CPU-bound busy loops to produce a current step that the power meter can detect in the current waveform.

Flow mode. Runs idle, CPU-bound, memory-bound and WiFi-transmit phases in sequence. Each phase runs for a configurable duration. This mode provides a repeatable multi-level load for calibration experiments.

Task mode. Runs a machine learning inference task N times at a configurable period. The firmware includes a ResNet-8 model trained on CIFAR-10 (int8 quantization, MLPerf Tiny benchmark). Two clock policies are available:

- Policy A: the CPU stays at 240 MHz throughout.
- Policy B: the CPU runs at 240 MHz during inference and drops to 80 MHz between tasks.

Policy A measures the energy of one inference at full clock speed. Policy B measures the energy of the complete task–idle cycle with frequency scaling.

In the first inference recordings, we noticed a small periodic burst which was still present when no inference should run. It came from the 802.11 beacon traffic and

was about 10 mA to 20 mA, unfortunately in the same region as part of the inference signature. Later firmware therefore disconnects WiFi before each run. This removes the interference, but it also means that task commands have to be prepared before the measurement starts.

The **marker agent** is a Python script on the server that commands the DUT to generate markers at the start and end of an experiment. The time synchronization pipeline (Subsection 5.4.2) uses these markers to compute the clock offset and drift between the server and the power meter.

5.3. Server and Data Recorder

We wrote the server in Python using FastAPI, with an embedded MQTT client (paho-mqtt). It runs on the same PC that hosts the MQTT broker (Mosquitto, introduced in the design chapter) and registers the mDNS hostname `thesis-hub.local` so that devices on the local network can find it without a static IP configuration. The server records all measurements to a database and provides REST endpoints for experiment control.

5.3.1. Data Storage

We store measurements in an SQLite database in write-ahead logging mode. The relevant database tables are summarized in Table 5.5.

Table	Key columns	Purpose
`samples`	`ts_ms`, `label`, `ma`, `mw`, `mwh`	One row per channel per monitoring cycle
`workload_events`	`ts_ms`, `workload`, `device`	Workload phase transitions (deduplicated)
`experiments`	`id`, `name`, `t_start`, `t_end`	Experiment metadata and time bounds
`experiment_steps`	`id`, `exp_id`, `step`, `rep`	Per-step windows within an experiment

Table 5.5.: SQLite database schema (key columns only).

The `samples` table receives one row per channel per monitoring message. At 10 Hz with three channels, this produces approximately 2.6 million rows per day. The table is indexed on `(ts_ms)` and on `(label,ts_ms)` for the two most frequent query patterns: time-range export and per-channel history.

5.3.2. MQTT Subscriptions

The server subscriptions are:

- `powermeter/+/telemetry`—monitoring messages from the power meter(s).
- `powermeter/+/cal`—retained calibration tables.
- `workload/s3/state`—the DUT's self-reported workload state.

Each monitoring message is parsed and inserted into the `samples` table. A network statistics module tracks per-device throughput, message rate, and the 50th/95th-percentile inter-message gaps over the last 200 messages.

5.3.3. Experiment Lifecycle

We delimit measurement windows through REST endpoints. The experiment scripts call them in order:

1. `POST /api/exp/start`—creates an experiment record with a name and optional metadata.
2. `POST /api/exp/step/start`—marks the start of one step (a workload level at a repetition number).
3. `POST /api/exp/step/end`—marks the end of the step.
4. `POST /api/exp/end`—closes the experiment.

Two query endpoints extract the recorded data: `GET /api/exp/{id}/summary` computes per-step, per-channel statistics (mean current, standard deviation, mean power, sample count), and `GET /api/exp/{id}/export.csv` exports the raw sample rows.

The orchestrator (`orchestrator.py`) automates the full sequence: it connects to the MQTT broker, starts an experiment, iterates over steps and repetitions (apply load, wait for the system to settle, tag the capture window, release the load), fetches the summary, and optionally controls the Monsoon supply voltage and safety limits.

5.4. Analysis Pipeline

The offline analysis is split across Python scripts for calibration, time synchronization and stream decoding. They share exported experiment data but run independently of each other.

5.4.1. Calibration

The calibration pipeline corrects systematic sensor errors against the Monsoon HVPM reference instrument.

The *fit stage* (`calibrate.py`) collects $(\text{reference},\ \text{sensor})$ current pairs from the per-step summaries of one or more calibration experiments. It fits the linear error model

$$I_{\text{raw}} = \mathit{gain} \times I_{\text{true}} + \mathit{offset}$$

by ordinary least squares and evaluates the worst-case relative current deviation before and after correction. Steps with fewer than five samples or with a reference current below a configurable threshold are excluded from the fit.

The *automation script* (`autocal.py`) combines the fit stage with experiment control. Its sequence is:

1. Verify that the server, the power meter, and the reference instrument are responsive (preflight).
2. Lock the sampling rate. A calibration is valid only at the rate it was fitted at, because the ADC configuration changes with the rate (Subsection 5.1.2).
3. Clear any existing calibration so the fit sees the raw sensor error, not a residual on top of an old correction.
4. Run the stepped load sequence N times through the orchestrator.
5. Compute the least-squares fit.
6. Write the gain and offset to the power meter's NVS.
7. Run an independent verification round and measure the residual error with the correction applied.

The verification round uses a fresh experiment that was not part of the fit. Its worst-case error is a measured quantity, not an in-sample statistic.

5.4.2. Time Synchronization

The time synchronization script (`marker_align.py`) implements the method from Subsection 4.6.2. It takes the current waveform from the power meter and locates the load markers placed by the marker agent.

The algorithm proceeds as follows:

1. **Template.** Build a normalized template of the expected marker pattern (100 ms, 200 ms, 100 ms bursts) at the waveform's sampling rate.
2. **Correlate.** Compute the normalized cross-correlation between the template and the waveform. The correlation peak locates the marker to within one sample period.
3. **Refine.** Fit a parabola to the three highest correlation values around the peak to obtain sub-sample precision (0.47 ms to 1.07 ms measured at 2 kHz).

When multiple markers are present (start and end of an experiment), the script fits a first-order drift model (offset + rate × elapsed time) and reports the residuals. The peak-to-second-peak ratio of each correlation peak (Subsection 4.6.3) serves as a quality metric: a ratio below a configurable threshold flags a false or ambiguous match.

5.4.3. Stream Decoder

The stream decoder (`stream_recv.py`) receives binary batches from the streaming capture and transition capture engines. It verifies the magic bytes and sequence numbers, unpacks the header and sample arrays according to Table 5.4, and produces an array of (timestamp, current) pairs. A gap in the sequence numbers indicates a lost batch. The decoder reports the gap so the consumer can decide whether to interpolate or discard the affected window.

5.5. Endpoint Summary

Table 5.6 summarizes all programmatic interfaces of the platform. The full REST endpoint specifications and the MQTT topic tree are given in the appendices.

Layer	Interface	Count	Scope
Device	REST	9	Monitoring readout, sensor scan, calibration, sampling rate, energy reset
Device	MQTT pub	5	Monitoring messages, calibration table, burst/streaming output, transition capture output, accumulator readout
Device	MQTT sub	1	Commands (rate, calibration, capture, accumulator, reset)
Server	REST	20	History, CSV export, workload control, DUT management, calibration, rate, experiment lifecycle
Server	MQTT sub	3	Monitoring messages, calibration tables, workload state
Server	MQTT pub	3	Workload commands, energy reset, rate change

Table 5.6.: Programmatic interfaces by layer.

6. Evaluation

The prototype looked stable in daily use, but this was not enough to claim a useful measurement platform. We therefore tested it from the calibration bench up to the final task-energy value. Some results were expected, such as the different sensor ceilings. Others were not: in particular, a configured 50 Hz stream was sometimes only delivered near 33 Hz. The experiments below include these unsuccessful operating points as well as the final ones.

6.1. Test Bench and Reference Instrument

All experiments use the same bench, although channels were connected and disconnected depending on the test. The power meter board (`pma01`) runs four sensor channels on a shared I^2C bus: one per DUT, each through a series shunt resistor (Table 5.1), plus an auxiliary INA228 used in the dual-chip comparisons. Table 6.1 lists the channel assignments.

Channel	Sensor	DUT	V_{bus}	Shunt
`0x40`	INA228	auxiliary (dual-chip and S3-supply experiments)	5 V	0.015 Ω
`0x41`	INA219	ESP32-S3-EYE	5 V	0.100 Ω
`0x44`	INA228	Jetson Orin Nano	19 V	0.015 Ω
`0x45`	INA226	Raspberry Pi 5	5 V	0.002 Ω

Table 6.1.: Test bench channel assignments.

Reference instrument. A Monsoon High Voltage Power Monitor (HVPM) serves as the external reference. It samples at 5 kHz and provides both current and voltage, but its input voltage is limited to 13.5 V. This limit excludes the 19 V Jetson supply from direct reference comparison. On the 5 V supplies, the HVPM is inserted in series with the power meter channel: HVPM(+) → sensor module → DUT → HVPM(−). Figure 6.1 shows this insertion on the ESP32-S3 supply; the wiring is the same as in Figure 4.3, and the only change is that the HVPM takes the place of the power supply, feeding the bus voltage and closing the return path through its own terminals. A

photograph of the physical setup is in Appendix C. The two instruments therefore share the same physical current but measure voltage at different nodes—the HVPM at its own terminals, the sensor at the DUT side of the shunt resistor. This difference is relevant when the series resistance between the two measurement points causes a voltage drop proportional to load current (Section 6.7).

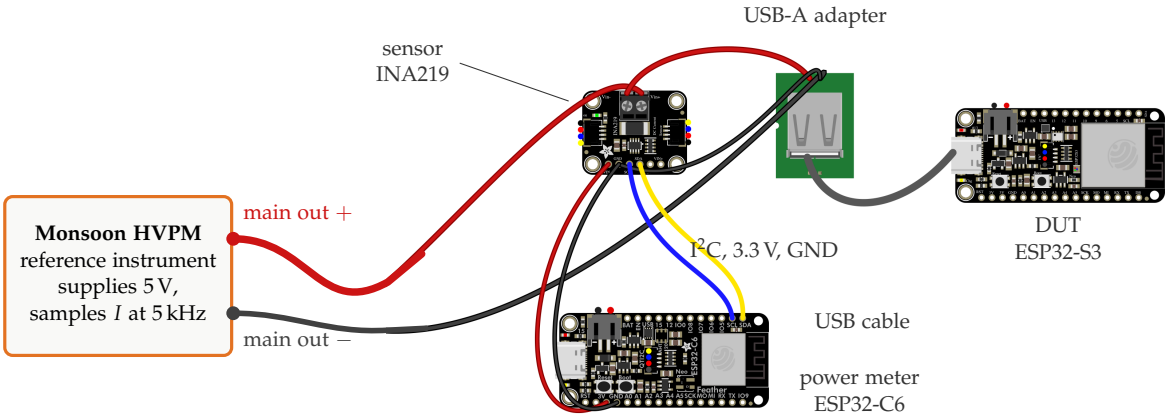


Figure 6.1.: Calibration wiring on the ESP32-S3 supply. The Monsoon HVPM replaces the external power supply: its main output + feeds the current path into the sensor's IN+ screw terminal, the DUT's ground returns to the main output −, and the HVPM samples the same current at 5 kHz. Everything to the right of the instrument is identical to Figure 4.3.

6.2. Calibration Accuracy

Each sensor channel is calibrated with a linear model (Equation 5.1). The calibration pipeline (Subsection 5.4.1) drives the DUT through a fixed stepped load sequence, records the reference current from the HVPM, and fits gain and offset by least-squares regression. The resulting coefficients are written to the power meter's NVS and verified in an independent round with the coefficients already applied.

Table 6.2 shows the calibration results for the four channels.

The INA226@0x45 channel was calibrated and verified on the 5 V Pi 5 supply. The INA228@0x44 coefficients were fitted and verified at a 12 V supply, because the reference instrument does not reach 19 V (Section 6.8); the INA228@0x40 and INA219@0x41 channels were calibrated on the 5 V S3 supply. The INA228@0x40 channel shows the largest verification residual. At its 86–200 mA operating points, the reference instrument's own low-range uncertainty is about ±1%, so part of this residual belongs

Channel	Gain	Offset (mA)	Before cal.	Fit residual	Independent verification
INA226@0x45	1.004140	0.790	0.58%	0.07%	0.10%
INA228@0x44	1.013195	4.469	1.98%	0.37%	0.08%
INA228@0x40	1.013054	3.826	5.12%	0.74%	1.7%
INA219@0x41	0.997887	4.136	3.60%	0.35%	0.23%

Table 6.2.: Calibration results per channel. “Before” and “verification” are worst-case residuals of load steps; the verification round runs with the coefficients applied, on data that was not part of the fit.

to the reference. Figure 6.2 shows the effect on the Pi 5 channel. Before calibration, every repetition lies 0.41–0.58% above the reference; after the coefficients are written to the firmware, the independent verification run stays within +0.10% over the whole load sequence.

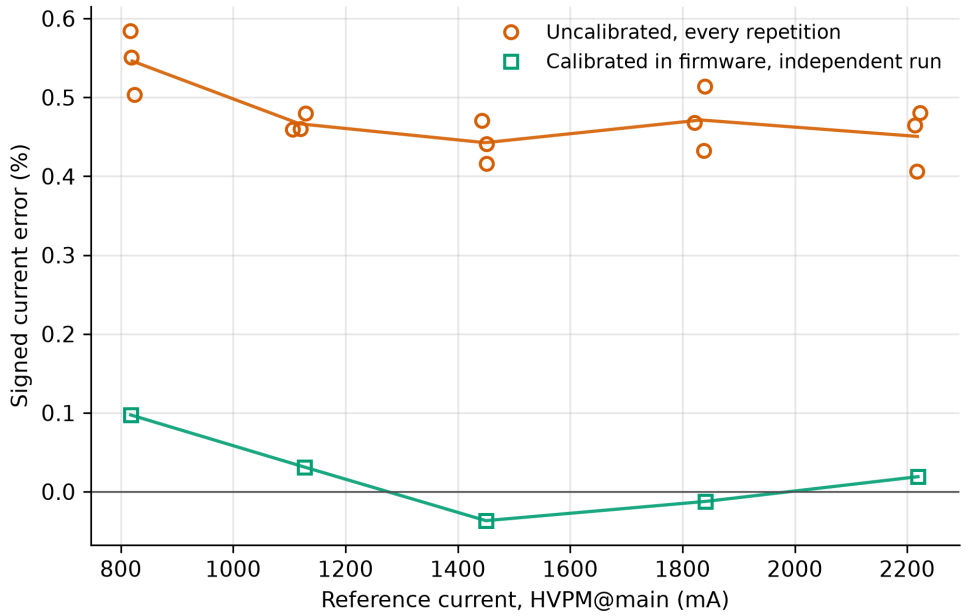


Figure 6.2.: Signed current deviation of the INA226@0x45 channel against the reference instrument (Pi 5 supply, 10 Hz). The calibrated points come from an independent verification run with the coefficients already applied in the firmware.

Uncalibrated, all points of the load sequence lie above the reference; after calibration, the residuals sit on both sides of zero, which means the systematic bias is removed and only noise-level scatter remains.

Some details behind these numbers are worth discussing.

Gain consistency across chips. The gain of the two INA228 modules differs by only 0.014% (1.013195 vs. 1.013054). Gain is a property of the chip and shunt resistor combination; the offset is a property of the individual module. This supports the argument that gain can be transferred across channels that use the same sensor model and shunt value, while offset cannot.

Coefficient stability over days. A calibration is only useful if its coefficients stay valid for some time. A first check used the S3 load sequence again, six days after the INA219 calibration, and gave a worst-case residual of 1.6%. But this bound was set by the workload, not by the sensor: two runs of the same sequence taken minutes apart already differ by a comparable amount, because the `mem` workload does not draw a repeatable current. We therefore repeated the check with more repeatable loads. On the Pi 5 supply, the INA226@0x45 coefficients—fitted nine days earlier—were verified again with three rounds of the load sequence over five core levels (505–2153 mA); Table 6.3 lists every point. Over all $n = 15$ points, the mean residual is $+0.05\%$ with a standard deviation of 0.19 percentage points; the worst single point is $+0.59\%$ (one idle step), and all other points stay within 0.3%. On the S3 supply, the INA228@0x40 residuals measured three days after calibration match the earlier values to within 0.30 percentage points across the idle, `cpu`, and `wifi_tx` workloads. The coefficients therefore hold at least over days. The practical limit of such a check is the repeatability of the load, not a drift of the coefficients.

Load level	Round 1	Round 2	Round 3
idle	$+0.59\%$	$+0.14\%$	$+0.05\%$
cpu1	$+0.12\%$	-0.02%	$+0.11\%$
cpu2	-0.02%	$+0.06\%$	$+0.10\%$
cpu3	$+0.07\%$	-0.05%	$+0.16\%$
cpu4	-0.20%	-0.08%	-0.28%

Table 6.3.: Signed current residuals nine days after calibration (INA226@0x45, Pi 5 supply, three independent rounds, 505–2153 mA).

Effect on the hardware accumulator. The INA228 energy accumulator integrates the raw, uncalibrated current. Before comparing it against the calibrated sampling integral, the readout must be corrected:

$$E_{\text{corrected}} = \frac{E_{\text{hw}} - V \cdot I_{\text{offset}} \cdot t}{\text{gain}} \tag{6.1}$$

where E_{hw} is the register readout, V the bus voltage, I_{offset} the calibration offset, t the elapsed time, and gain the calibration gain.

6.3. Transport Performance

We next compare the per-message cost of the transports and check what happens when more consumers connect. This was measured on the device, not inferred from packet sizes.

6.3.1. Per-Message Transmission Cost

The firmware can transmit each monitoring message through MQTT (QoS 0, publish), HTTP POST (new TCP connection and keep-alive), WebSocket, or raw UDP. Table 6.4 lists the per-message cost measured on the device, with the three-channel monitoring payload of approximately 640 B.

Mode	t_{p50}	t_{p95}	t_{max}	Max. rate at 50 Hz
MQTT QoS 0	0.72 ms	1.3 ms	2382 ms	50 Hz
HTTP POST (new conn.)	84 ms	104 ms	161 ms	11.5 Hz
HTTP POST (keep-alive)	22 ms	83 ms	1170 ms	34–37 Hz
WebSocket	1.2 ms	1.5 ms	2.1 ms	50 Hz
UDP	0.70 ms	0.92 ms	1.7 ms	50 Hz

Table 6.4.: Per-message transmission cost (device-side), 48 runs, zero loss, zero reordering. t_{p50} is the median across runs; t_{p95} and t_{max} are the worst run's 95th percentile and the overall maximum. The second-scale maxima are transmission stalls; Subsection 6.4.1 discusses their effect on the delivered rate.

MQTT and WebSocket deliver all messages at 50 Hz in this benchmark, with median per-message costs below 2 ms; the isolated second-scale maxima on the MQTT and keep-alive rows are network stalls, not per-message cost. This 50 Hz figure is an upper bound under favorable network conditions: the firmware pauses sampling while a transmission is blocked, so the sustained monitoring rate in a real network stays below

the configured 50 Hz (Subsection 6.4.1). HTTP POST with a new TCP connection per message costs approximately 84 ms—two orders of magnitude more—and cannot sustain rates above approximately 11.5 Hz. Keep-alive connections reduce the median to 22 ms but exhibit a bimodal distribution: the p95 of 83 ms corresponds to the periodic TCP re-establishment. The platform uses MQTT for the continuous data path (Subsection 4.4.4); the HTTP interface serves infrequent control requests.

6.3.2. Subscriber Scalability

A dedicated experiment measures the device-side cost of concurrent consumers. The monitoring rate is fixed at 10 Hz. For each $N \in \{1, 2, 4, 8\}$, N independent subscribers connect to the broker (MQTT group) or N independent pollers query `/api/v1/power` at 1 Hz (REST group). Each run lasts 120 s. Table 6.5 shows the results.

Group	N	t_{acq}	t_{pub}	Device Hz	Consumer p95	Failures
MQTT	1	5.964 ms	4.680 ms	10.00	—	0
MQTT	2	5.964 ms	4.694 ms	9.88	—	0
MQTT	4	5.964 ms	4.688 ms	9.98	—	0
MQTT	8	5.964 ms	4.689 ms	10.00	—	0
REST	1	5.963 ms	4.711 ms	10.00	99 ms	0
REST	4	5.963 ms	4.707 ms	9.40	133 ms	4
REST	8	5.964 ms	4.711 ms	9.48	1084 ms	54

Table 6.5.: Device-side cost vs. number of consumers. "—" = not applicable: MQTT subscribers send no requests, so they have no request latency.

In the MQTT group, the device publishes one message per sampling cycle regardless of N. Fan-out is handled by the broker. The acquisition and publish times are identical across all four values of N.

In the REST group, the cost does not appear in the per-cycle timing (the HTTP server runs outside the timed acquisition and publish phases) but in consumer-side quality of service. At $N = 8$, the polling latency p95 rises from 99 ms to 1084 ms, and 54 of the 968 scheduled requests (5.6%) fail. Figure 6.3 plots both sides of the same experiment: the device-side cycle cost stays flat from one to eight consumers, while the share of served REST reads drops to about 94% at $N = 8$.

6.3.3. End-to-End Latency

End-to-end latency has two components: the *sample age* (time from sensor read to arrival at a subscriber) and the *closed-loop latency* (time from a physical event to a

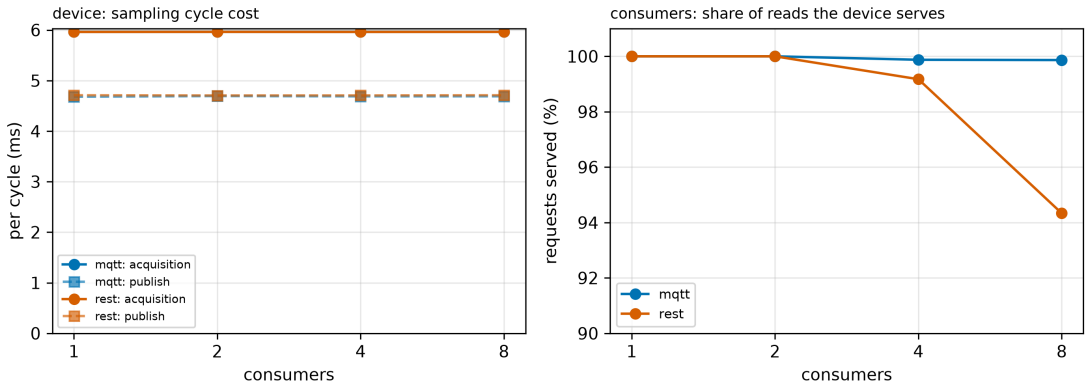


Figure 6.3.: Device-side sampling cycle cost (left) and share of consumer reads served (right) for one to eight concurrent consumers at 10 Hz. The acquisition and publish times do not depend on the number of consumers in either group; the REST cost appears on the consumer side.

software-initiated response reaching the DUT).

Metric	p50	p95	Conditions
Sample age	237 ms	564 ms	10 Hz, three channels
Closed-loop latency	490 ms	890 ms	10 Hz, cpu workload

Table 6.6.: End-to-end latency components.

The closed-loop latency was measured using an automated feedback loop: a subscriber detects a workload transition in the incoming monitoring data (two consecutive messages crossing a threshold), publishes the reverse command, and a simultaneous HVPM capture records both physical edges. For the same runs, the component medians are: sample age 221 ms, two-message confirmation wait 65 ms, software response <1 ms, and DUT execution 261 ms (component medians do not sum to the median of the total). The platform's own contribution (sample age plus confirmation wait) is about 286 ms.

6.4. Delivered Sampling Rate

The firmware accepts configured sampling rates from 1 Hz to 50 Hz for averaged monitoring and from 100 Hz to 3000 Hz for burst and streaming capture. Transition capture has a practical limit of 2 kHz and an absolute limit of 2.4 kHz (as described in

the implementation chapter). A configured value is not necessarily the rate arriving at server, so both are reported.

6.4.1. Averaged Monitoring

A controlled experiment varies the configured rate and the deadline policy (assigning `now` vs. accumulating the next deadline). Each cell runs for 30 s with two repetitions. Table 6.7 shows the results for the three-channel configuration.

Config. (Hz)	Deadline	Delivered (Hz)	Period p99 (ms)	Late (%)
20	assign	19.3–20.0	52	0.09
20	accum.	**20.00**	52	**0.00**
50	assign	39.0–42.3	21	0.3
50	accum.	32.5–44.1	21–151	2.9

Table 6.7.: Delivered rate vs. configured rate (three channels, display off, two repetitions per cell).

Up to 20 Hz, the delivered rate follows the configured rate: with the accumulating deadline policy, the firmware delivers exactly 20.00 Hz with zero late cycles and zero stalls.

Per-cycle budget. A single sampling cycle at 50 Hz requires: acquisition 5.95 ms + publish 4.51 ms = 10.47 ms. This fits within the 20 ms period with 9.5 ms of margin. The rate deficit at 50 Hz is therefore not caused by per-cycle processing.

Display refresh. The clearest single cause of late cycles is the onboard display. One LCD refresh occupies the single core for about 70 ms (Subsection 5.1.4), so a refresh that lands near a sampling deadline delays that cycle: at 20 Hz with the display on, the delivered rate drops to 17.3–19.8 Hz with about 10% late cycles. With the display off, delivery at 20 Hz is exact, which is why the rate experiments run with the display off.

50 Hz: transmission pauses. At the 50 Hz setting, the delivered rate is 32.5–44.1 Hz. The cause is the network path: while a transmission is blocked, the firmware pauses sampling until the send completes. Each 30 s window loses 3.6 s–10.3 s to such pauses (approximately four per window, 1.15 s each), and the same mechanism appears during streaming capture (Subsection 6.4.2). How often a send blocks depends on the network

environment, so the configured 50 Hz is an upper bound that holds only when the network does not block; the delivered rate at this setting must be measured per session.

6.4.2. Burst and Streaming Capture

The maximum sampling rate depends on the sensor model, the I^2C bus clock, and the acquisition mode. Table 6.8 shows the achieved rates in burst capture (RAM-buffered) across three bus clock speeds.

Sensor	100 kHz	200 kHz	400 kHz	New-conv. rate
INA219	905 Hz	1616 Hz	2106 Hz	∼895 Hz
INA226	906 Hz	1619 Hz	2673 Hz	∼2373 Hz
INA228	778 Hz	1412 Hz	2378 Hz	∼2355 Hz

Table 6.8.: Maximum achieved sampling rate in burst capture (2 s per point).

At 100 kHz, the bus transfer time (∼1.1 ms per sample) is the bottleneck. At 400 kHz, the INA219 still reads at 2106 Hz, but its new-conversion rate saturates at its 532 µs conversion time (∼895 Hz): reads above this rate return duplicates of the previous conversion. The new-conversion rates of the INA226 and INA228 continue to scale with bus speed. Below 500 Hz, all three sensors follow the requested rate exactly with zero sample loss.

Streaming endurance. Streaming capture publishes binary batches over MQTT during acquisition. Table 6.9 shows the results at 2 kHz with the INA228 at 400 kHz bus clock.

Request (Hz)	Duration (s)	Samples	Completeness (%)	Payload (kbps)
1000	20	20 000	100.0	62
2000	20	40 000	100.0	124
2000	120	240 000	100.0	128
2000	120	230 671	96.1	123
2400	20	44 856	93.5	140
2400	60	135 152	93.9	143

Table 6.9.: Streaming capture completeness (INA228, 400 kHz bus). The two 120 s rows are the best and worst of five runs.

At 2000 Hz, 20-second captures are complete; over 120 s, sampling completeness is 96.1–100% across five runs, while network delivery stays at 100% throughout, with a

payload of about 125 kbit s^{-1}—less than 1% of the WiFi link capacity. The transport is not the bottleneck. At 2400 Hz, the sensor read time ($\sim$0.42 ms) approaches the 0.42 ms sampling period, and completeness drops to 94%. The practical ceiling for complete streaming capture is therefore 2000 Hz at the tens-of-seconds scale.

Batch size. A batch size ablation (50, 100, 200, 480 samples per batch) at 2000 Hz confirms that batch size does not affect sampling completeness: seven of eight runs achieve 100%. The per-publish pause increases from 1.82 ms to 2.40 ms as batch size grows, but its frequency decreases proportionally. The single incomplete run (96.4%) is caused by a 1255 ms network stall, not by the batch size.

6.5. Transition Capture

Transition capture arms the sensor at a practical 2 kHz (absolute limit: 2.4 kHz) and captures a window around the first current transition that exceeds the configured threshold. This section validates the mechanism against both self-tests (no reference instrument) and reference-instrumented measurements.

6.5.1. Self-Test

The ESP32-S3 supply serves as the test case. The threshold is set empirically: a 900 s (15-minute) idle scan at 900 Hz measures the maximum moving-mean current over a 150 ms window. The result is 170.3 mA; the threshold is set to 200 mA, providing 29.7 mA (14.8%) of margin.

Trigger reliability. Three repetitions with a `wifi_tx` workload trigger successfully (3/3). Each capture contains 1080 samples (180 pre-trigger + 900 post-trigger), all received without batch gaps or publish failures.

Zero false triggers. The same 900 s idle scan doubles as the false-trigger check: over its 810 079 samples, zero trigger events occur, and the maximum moving mean stays 14.8% below the 200 mA threshold.

6.5.2. Reference-Instrumented Validation

A simultaneous HVPM capture (5 kHz, 4 s per round) provides an independent view of the same physical current. $N = 20$ rounds, each triggered by a `wifi_tx` command. All 20 rounds trigger successfully. Figure 6.4 shows one captured round: the power

meter's 900 Hz trace follows the 5 kHz reference through the current transition, and the trigger fires 86 ms after the transition, as the sustain window requires.

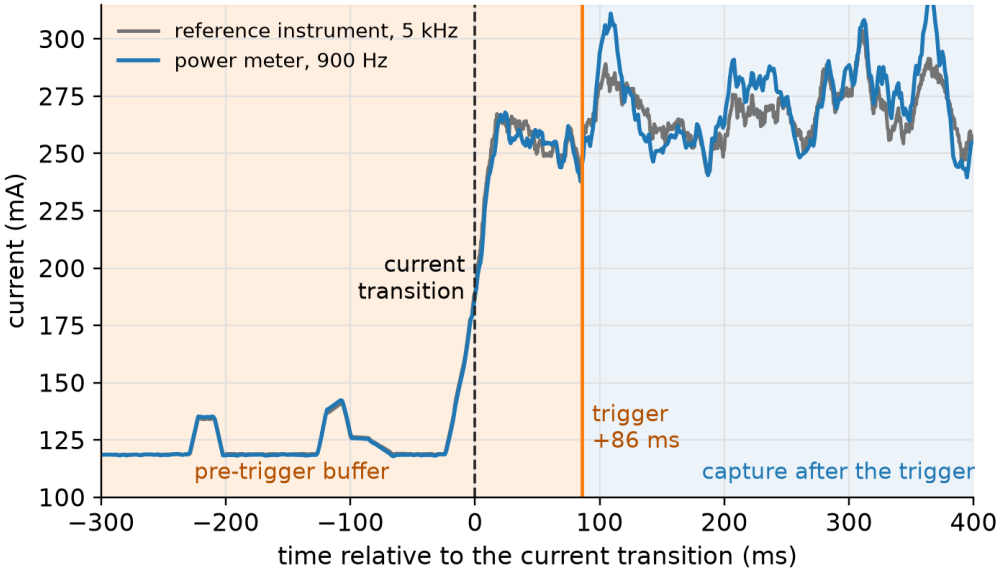


Figure 6.4.: One transition capture on the ESP32-S3 supply: power meter (900 Hz) and reference instrument (5 kHz) on the same power input, 20 ms moving mean. The shaded regions are the pre-trigger buffer and the capture after the trigger.

Current levels. Table 6.10 compares the idle and loaded current levels measured by the two instruments on the same series circuit.

Level	Reference	Meter (INA219)	Ratio (med. ± sd)
Idle	121.9 mA	121.3 mA	0.996 ± 0.002
Loaded	260.0 mA	260.8 mA	1.000 ± 0.012
Step	138.1 mA	139.5 mA	1.004 ± 0.022
p99 peak	453.2 mA	403.0 mA	0.890 ± 0.056

Table 6.10.: Current levels: power meter vs. reference (ESP32-S3 supply, $n = 20$).

From Table 6.10 we can see that the idle, loaded, and step levels agree within 0.4%. The p99 peak shows an 11% deficit in the power meter reading; this is a consequence of the INA219's 532 µs conversion time, which acts as a low-pass filter on sub-millisecond

transients (Subsection 6.5.3).

Confirmation delay. The time from the physical edge (50% crossing point of the smoothed reference trace) to the trigger event is: median 85.1 ms, p95 123.8 ms, sd 10.8 ms. This delay is the time the moving mean needs to cross the threshold: starting from the idle level $\mu_0 = 121.9\,\text{mA}$ and stepping to $\mu_1 = 260.0\,\text{mA}$, the moving mean over the sustain window $S = 150\,\text{ms}$ reaches the threshold $\theta = 200\,\text{mA}$ after $t = S\,(\theta - \mu_0)/(\mu_1 - \mu_0) = 84.8\,\text{ms}$—within 0.3 ms of the measured median.

On the high-contrast Jetson supply (>2× step), the same firmware with a 5 ms sustain window achieves a confirmation delay of only 5.71 ms (median, $n = 20$).

Host-timestamp alignment error. Aligning the two instruments by host timestamps (MQTT publish time for the power meter, block return time for the HVPM) introduces a systematic error of $-234\,\text{ms} \pm 27.6\,\text{ms}$ relative to signal-based alignment. This exceeds the confirmation delay itself and confirms that transition capture inside the firmware avoids the command-path jitter that host-side segmentation would introduce.

6.5.3. Effect of Conversion Time on Peaks

To isolate the cause of the 11% peak deficit, a three-instrument experiment places both an INA228@0x40 and the INA219@0x41 on the same series circuit, both sampling at 2000 Hz on a 400 kHz bus. The only difference between the two sensors at the same sampling rate is their per-sample conversion time: 50 µs (INA228) vs. 532 µs (INA219).

	INA228 / Ref	INA219 / Ref
`wifi_tx` p99 ratio	1.14	0.88

Table 6.11.: P99 current ratio to the reference (5 kHz HVPM), `wifi_tx` workload. All three p99 values are computed over the common 18 s time window of the three instruments; across three rounds, the INA228 ratio varies between 1.01 and 1.14.

This is a distribution-level comparison: the three instruments sample the same current at different instants, so the p99 values describe their amplitude distributions, not the same individual peaks. The INA228 reports a p99 *above* the 5 kHz reference, despite sampling at only 2 kHz. This is explained by the aperture difference: the reference's 200 µs sampling period averages over a longer window than the INA228's 50 µs conversion. The INA219 shows a deficit at 2 kHz (0.88) just as at 900 Hz in the

trigger validation (0.890), confirming that the deficit is determined by per-sample conversion time, not by sampling rate.

6.5.4. Reporting Policy Comparison

An offline experiment applies threshold reporting (send-on-delta) to a recorded 10 Hz trace of 1175 monitoring messages covering five workload phases. Table 6.12 shows the trade-off between message reduction and reconstruction error.

Δ (mA)	Messages sent	Fraction (%)	Energy error	Max current error (mA)
0.5	1015	86.4	+0.24%	0.49
5	757	64.4	+0.21%	4.93
20	119	10.1	+0.03%	19.84
50	25	2.1	+2.29%	49.99

Table 6.12.: Threshold reporting: message count vs. reconstruction error (zero-order hold).

At $\Delta = 20$ mA, the message count drops to one-tenth while the energy integral error is only 0.03%. However, the instantaneous current error reaches 19.84 mA—16% of the 124 mA idle baseline. The trigger threshold design (Subsection 4.5.2) relies on waveform features (burst amplitude, duration) that would be destroyed by this level of reconstruction error.

All in all, the data reduction strategies serve different purposes: threshold reporting reduces the message volume at all time points but degrades waveform fidelity; transition capture retains full resolution at events and produces no data between them.

6.6. Time Synchronization

Offline time synchronization, described above in the implementation, corrects the clock offset and drift between the power meter and the host by cross-correlating load markers. This section reports the residual alignment error after correction.

6.6.1. Alignment Residuals

Figure 6.5 shows what a located marker looks like. The grey bands are the pulse intervals that the DUT reports in its own clock; after the alignment, the pulses in the power meter's 400 Hz stream and in the 5 kHz reference trace fall exactly into these

bands. In the 20 Hz monitoring stream (right panel), the same pulses are smeared over a few samples—a slow stream still finds the marker, but it cannot place it precisely.

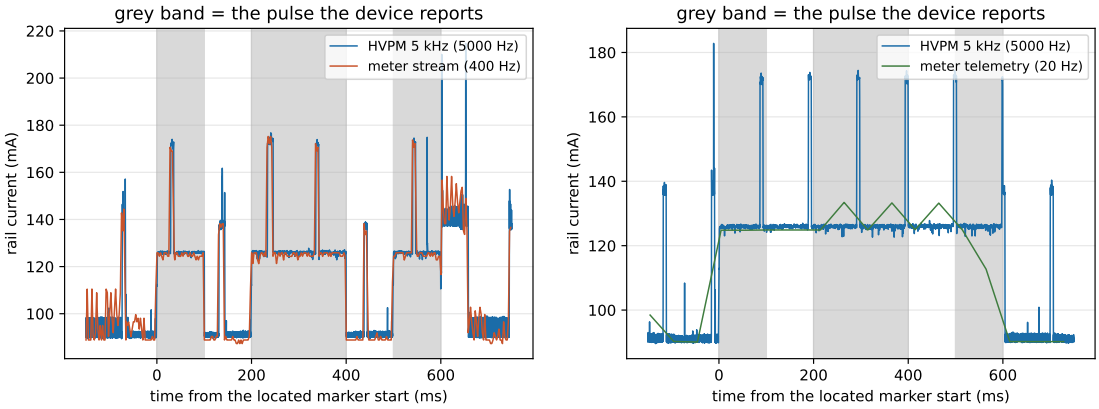


Figure 6.5.: One load marker after alignment, seen by three instruments on the same supply. Grey bands: the pulse intervals reported by the DUT in its own clock. Left: the reference instrument (5 kHz) and the power meter's streaming capture (400 Hz). Right: the same marker in the 20 Hz monitoring stream.

How precisely a marker can be placed depends on the sampling rate. Figure 6.6 collects the location error of all markers: the median falls from roughly 100 ms at 10 Hz monitoring to 0.13 ms at 2 kHz streaming capture, and the 95th percentile at kilohertz rates is 0.5–1.1 ms. This measured curve is the basis for the two-level design in Subsection 4.6.2: the monitoring stream is good enough for coarse alignment, and the fine level needs the kilohertz streaming capture.

Table 6.13 shows the alignment residuals across multiple runs, separated by instrument pair. The stream channel is aligned between the power meter and the host; the HVPM channel is aligned between the HVPM and the host.

Pair	Resid. med (µs)	Resid. p95 (µs)	Resid. sd (µs)	Drift (ppm)	Pulses used
Stream–host	132	1066	359	12.9	106
HVPM–host	147	650	191	−1518	90

Table 6.13.: Time synchronization residuals after drift correction (best six-block run).

The stream–host alignment has a median residual of 132 µs and a drift of 12.9 ppm, which reflects the SNTP-disciplined clock on the power meter. The HVPM–host

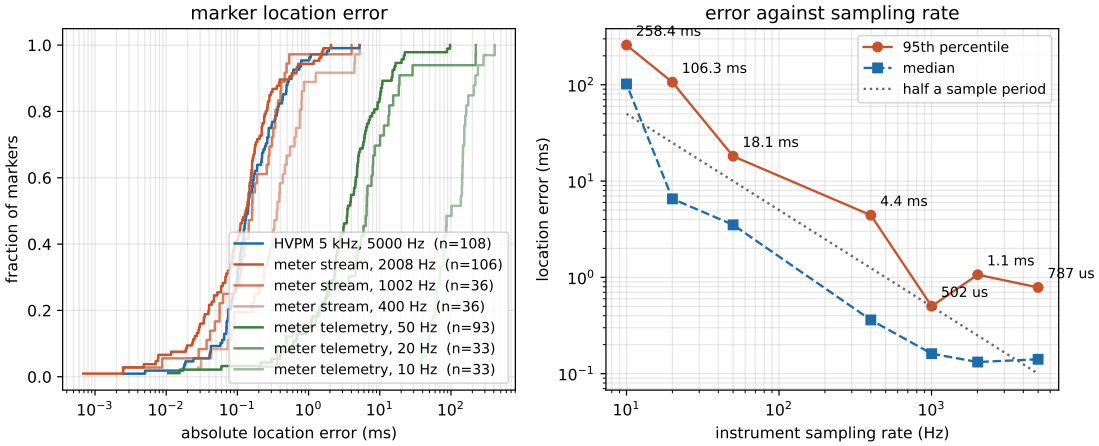


Figure 6.6.: Marker location error over all runs. Left: distribution per instrument and rate. Right: median and 95th percentile against the sampling rate; the dotted line is half a sample period, the limit of a discrete cross-correlation without interpolation.

pair shows a higher drift (−1518 ppm) because the HVPM's crystal oscillator is not synchronized to the network. After first-order drift correction, the residual median (147 µs) is comparable to the stream pair.

The first-order drift model itself is justified by a dedicated one-hour run (Figure 6.7): the offset between the two clocks grows linearly over the full 59 minutes. The drift rate is a per-run quantity—5.7–7.6 ppm in this run against 12.9 ppm in the six-block run of Table 6.13—which is exactly why the pipeline fits it from the markers of each run instead of reusing a stored value.

This run also checks the start-and-end marker design of Subsection 4.6.2 over a long duration. The run carried one marker per minute (60 markers, of which 49 pass the qualification gate), so the two-marker estimate can be compared against a fit through all of them: the drift rate from the first and the last marker alone differs from the all-marker fit by only 1.9 ppm, within the uncertainty of the two-point estimate itself. In terms of window error at 50 Hz, one hour without rate correction leaves a median error of 19.4 ms; the two end markers reduce it to 9.8 ms, and all markers to 4.1 ms, which is already the 50 Hz location error floor. The marker-based error therefore does not grow with the run duration. The SNTP-only error does: the right panel of Figure 6.7 shows that relying on the wall clock alone leaves an offset around 61 ms that wanders over a 48 ms range and grows by about 27 ms per hour—two orders of magnitude above the marker-based residual, and far outside the error budget of Table 4.6.

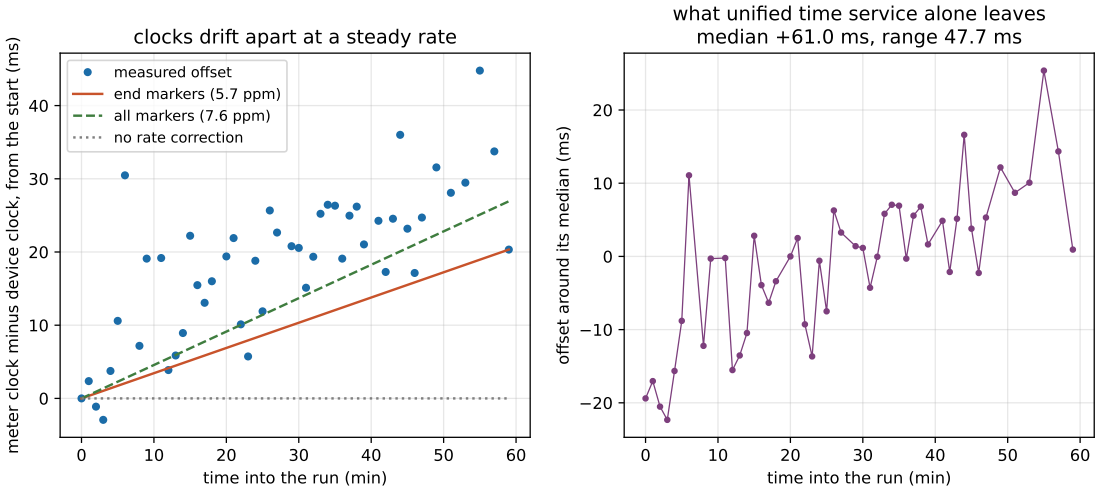


Figure 6.7.: Clock drift over a dedicated one-hour run. Left: the measured offset between the power meter clock and the DUT clock grows linearly (5.7–7.6 ppm, depending on which markers enter the fit). Right: the offset that remains when only the SNTP wall clock is used, without load markers.

6.6.2. Impact on Energy Integration

The boundary sensitivity test (Section 6.7) provides a direct measure of alignment error impact: shifting the integration boundary by 0.5 ms (one sampling interval at 2 kHz) changes the per-event energy by a median of 0.19%, with a maximum of 1.60%. Because the alignment residual (median 132 µs) is smaller than one sampling interval (500 µs), the alignment error contributes less than the boundary sensitivity to the energy measurement.

6.7. Task Energy Accuracy

This section measures the accuracy of per-task energy attribution by comparing the power meter's sampling integral against the reference instrument on the same series circuit. Two categories of experiments are presented: per-inference energy on a single DUT (Subsection 6.7.1), and cross-device current residuals after separating voltage and current contributions (Subsection 6.7.2).

6.7.1. Per-Inference Energy

The Jetson Orin Nano runs a FasterRCNN-MobileNetV3-Large-FPN object detector with one inference every 500 ms, with 20 warm-up inferences before the measurement window opens. The INA228@0x44 channel samples at 2 kHz (streaming capture); the INA228 hardware accumulator integrates concurrently. In separate runs, transition capture records individual events for comparison.

Quantity	Value
Idle baseline power	6.573 W (343.6 mA)
Per-inference energy (streaming capture, $n = 108$)	median 1.143 J, IQR [1.101, 1.270] J
Per-inference energy (transition capture, $n = 20$)	median 1.173 J
Device-side latency ($n = 130$)	median 216.0 ms (172 to 252 ms)
Boundary sensitivity (± 0.5 ms)	median 0.19%, max 1.60%

Table 6.14.: Per-inference energy on the Jetson Orin Nano (INA228@0x44, 19.13 V).

It can be seen from Figure 6.8 that the power trace inside one inference is far from flat: the 2 kHz streaming capture resolves the load changes within the 216 ms window, and the distribution of the per-inference energy has a long tail up to 1.86 J. A 10 Hz monitoring stream would average all of this structure away.

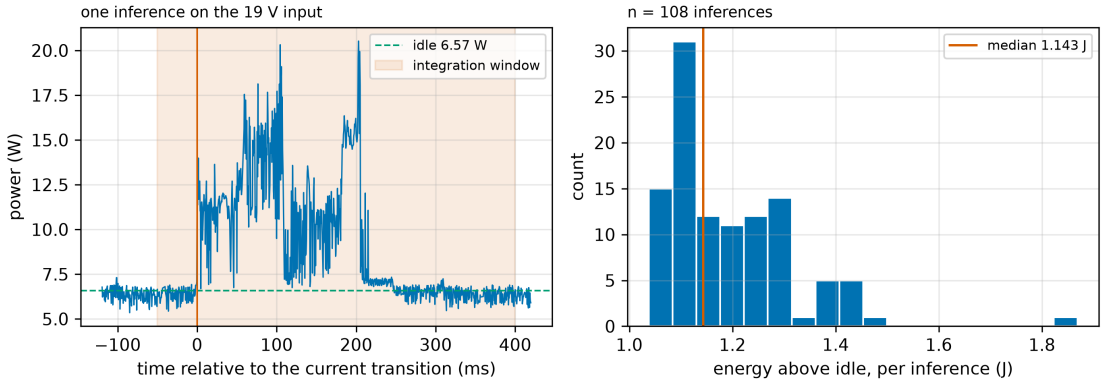


Figure 6.8.: One inference on the Jetson Orin Nano at the 19 V power input (left; the shaded region is the integration window, the dashed line the idle baseline) and the distribution of the energy above idle per inference for $n = 108$ inferences (right).

Three-way consistency. The strongest result of this evaluation is the agreement between three independent integration paths on the same data:

Comparison	Task period	Difference
2 kHz sampling vs. HW accumulator	500 ms	+0.076%
2 kHz sampling vs. HW accumulator	5 s	+0.007%
Accumulator (uncorrected) vs. sampling	500 ms	−2.15%

Table 6.15.: Three integration paths (Jetson supply).

After calibration correction, the independent software and hardware integrators agree within 0.08%, indicating no measurable aliasing loss. The uncorrected bias matches channel 0x44's gain and offset.

Data reduction. Transition capture records only the event window (550 ms), while streaming capture records the full measurement period. At a 500 ms task period, the trigger window exceeds the period, so transition capture produces slightly more samples than streaming capture. At a 5 s task period, transition capture records 11.6× fewer samples while preserving the energy result to within 1.4%. The data reduction factor scales with the ratio of the task period to the capture window; it is not a fixed constant.

6.7.2. Cross-Device Current Residuals

On the 5 V supplies, the HVPM and the power meter are in series but measure voltage at different nodes. The voltage difference between the two measurement points is proportional to the current through the series resistance (shunt resistor, wiring, connectors). To isolate the current measurement accuracy, the total energy difference is decomposed into a voltage component and a current residual:

$$\frac{E_{\text{meter}}}{E_{\text{ref}}} = \underbrace{\frac{V_{\text{meter}}}{V_{\text{ref}}}}_{\text{voltage ratio}} \times \underbrace{\frac{I_{\text{meter}}}{I_{\text{ref}}}}_{\text{current residual}}$$

Table 6.16 shows the results on the measured DUTs after applying this decomposition. For the ESP32-S3, the per-sample bus voltage is used, because this supply voltage droops under load.

Raspberry Pi 5. It can be seen that the −2.82% energy difference is almost entirely explained by the voltage ratio (−2.92%). The series resistance between the two voltage

DUT	Channel	Energy diff.	Voltage ratio	Current residual
ESP32-S3	INA228@0x40	−1.47%	−0.37%	−1.10%
Raspberry Pi 5	INA226@0x45	−2.82%	−2.92%	+0.10%
Jetson (12 V)	INA228@0x44	−1.01%	−0.87%	−0.14%
Jetson (12 V, rep 2)	INA228@0x44	−0.94%	−0.88%	−0.06%

Table 6.16.: Cross-device energy and current residuals.

measurement points is 0.178 Ω (measured under a four-core busy loop at 1847 mA). After subtracting the voltage component, the current residual is +0.10%.

Jetson Orin Nano. The Jetson's 19 V supply exceeds the HVPM's input range. A 12 V adapter supply enables a direct comparison. Across two independent runs, the total energy difference is −1.01% and −0.94%; the current residuals after voltage correction are −0.14% and −0.06%. Per-inference marginal energy (above baseline) is 1309 mJ (meter) vs. 1335 mJ (reference) in round 1.

ESP32-S3. On this supply, the residual (−1.10%) is of the same size as the reference instrument's own uncertainty: at ∼100 mA, the HVPM's low-range accuracy is about ±1%, so the comparison cannot attribute the difference to either instrument. The Pi 5 supply at ∼1 A is the operating point where the comparison has full discriminating power.

6.7.3. Hardware Accumulator Validation

The three-instrument experiment on the ESP32-S3 supply closes the evidence chain from external reference through sampling integral to hardware accumulator:

Workload	Accum. (corrected)	Monitoring integral	Diff.
idle	0.6138 W	0.6161 W	−0.37%
cpu	0.8269 W	0.8261 W	+0.09%
wifi_tx	1.3025 W	1.3040 W	−0.12%

Table 6.17.: Accumulator validation against external reference (ESP32-S3 supply, INA228@0x40).

The corrected accumulator agrees with the monitoring integral to within 0.37%. Without correction, the idle workload over-reports by 4.0%—the offset (3.83 mA) is 3.1%

of the 125 mA idle current. This confirms that the calibration correction is essential for the accumulator, and its relative impact is largest at low currents.

6.8. Limitations

The most visible limitation is the delivered rate at 50 Hz: the firmware pauses sampling while a transmission blocks (Subsection 6.4.1), and how often that happens depends on the network environment. The delivered rate at this setting must therefore be measured per session rather than assumed. At 20 Hz and below, the delivered rate matches the configured rate exactly.

Besides, when the DUT transmits heavily on the same WiFi channel, the power meter's streaming capture suffers heavy sample loss. In one experiment, the `wifi_tx` workload reduces streaming completeness to 13.9%, while the transition capture (which does not publish during the capture window) keeps the sample interval at its nominal 1.1 ms. This limits streaming capture to workloads that do not generate heavy WiFi traffic during the measurement window.

Reference and voltage-path limits. The HVPM's 13.5 V limit requires direct Jetson comparison at 12 V; at 19 V, validation is limited to the sampling integral's 0.076% agreement with the hardware accumulator. Per-channel calibration corrects current only, while bus voltage relies on factory calibration. On the Pi 5, the 2.92% voltage difference from the HVPM is explained by the series IR drop and remains uncorrected, so power derived from $V \times I$ inherits it.

INA226 has no hardware accumulator. The Pi 5 channel (INA226@0x45) does not provide a hardware energy register. Task energy on this supply can only be computed from the sampling integral, without the independent cross-check that the INA228's accumulator provides on the other two channels. Moving the INA228 to this supply is not practical: its 0.015 Ω shunt limits the full-scale current to 2.73 A, which leaves insufficient margin for the Pi 5's observed 1.85 A peak and startup inrush.

Single-run duration. The 32-bit `mono_us` timestamp wraps after 71.6 minutes, so longer experiments must be split into separately aligned runs. The one-hour drift run of Figure 6.7 stays within this limit; alignment beyond the hour scale remains untested.

7. Case Study: Offloading Energy

So far the power meter itself was the object of study. We now use it for the question that motivated the work: if two devices can execute the same inference, which device actually uses less energy?

The answer depends on more than the single-inference cost. Each offloading round incurs a fixed overhead (model loading, runtime initialization, OS scheduling) that must be amortized over the number of inferences per round. We therefore estimate the overhead and derive the point above which offloading becomes energy-efficient.

The measurements cover the ESP32-S3 microcontroller (5 V, INA228@0x40), the Raspberry Pi 5 (5 V, INA226@0x45), and the Jetson Orin Nano (19 V, INA228@0x44). The ESP32-S3 and the Pi 5 run the same ResNet-8 image classification model (MLPerf Tiny, 12.5 million multiply-accumulate operations per inference); the Jetson runs a FasterRCNN object detector as a representative GPU workload. Figure 7.1 shows the setup at one glance.

7.1. Single-Inference Energy

Table 7.1 lists the per-inference energy for each device, measured with the duty-cycle method described in Section 2.6.

Device	Model / runtime	Latency	Energy	Idle
ESP32-S3	ResNet-8 / C firmware	1406 ms	173.2 mJ	0.44 W
Pi 5	ResNet-8 / LiteRT (XNNPACK)	0.28 ms	0.55 mJ[†]	3.9 W
Jetson Orin Nano	FasterRCNN / PyTorch + CUDA	216 ms	1143 mJ	6.6 W

[†]Marginal energy per inference (slope of the fit over four batch sizes, Section 7.2).

Table 7.1.: Per-inference energy across three devices.

ESP32-S3. The ESP32-S3 takes 1406 ms per inference and consumes 173.2 mJ above the idle baseline. This is the slowest and, per inference, the second most energy-costly device.

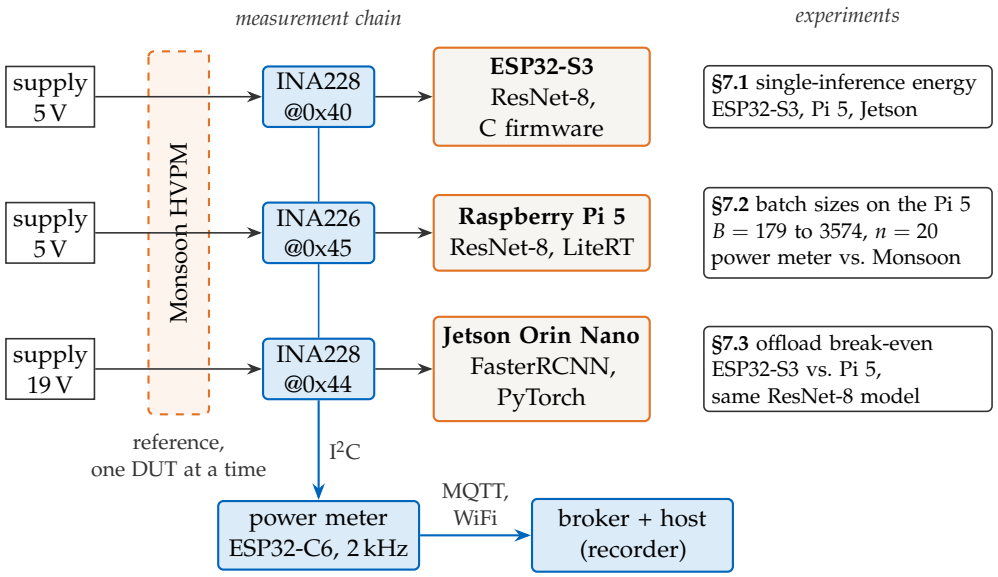


Figure 7.1.: Case study setup. The three DUT supply paths share the power meter's I^2C bus and MQTT link. The Monsoon HVPM is inserted in one supply path at a time as the reference; the right column lists the experiments.

Raspberry Pi 5. A single inference takes only 0.28 ms on the Pi 5 with the XNNPACK delegate—shorter than the power meter's 0.5 ms sampling interval at 2 kHz. The per-inference energy cannot be measured directly as a single event. Instead, the batch experiment in Section 7.2 derives the marginal energy from the slope of a linear fit: 0.55 mJ per inference. This is roughly 300 times less than the ESP32-S3 for the same model.

Jetson Orin Nano. The Jetson runs a different, larger model (FasterRCNN) that is not executable on the ESP32-S3. At 216 ms per inference and 1143 mJ per event (Table 6.14), the GPU workload is measurable as individual events.

7.2. Batch Size and Marginal Cost

The Pi 5's sub-millisecond inference time means that each offloading round can process many inferences. A round could also carry a fixed overhead—for example, the first inference after an idle period takes 0.54 ms against 0.28 ms in steady state.

To separate a possible fixed overhead from the per-inference marginal cost, four batch sizes are measured: 179, 715, 1787, and 3574 inferences per round. Each batch runs

20 times, one round every 3 s. The energy above the idle baseline uses one estimator for all runs: the baseline is the mean of all samples outside the task band (the task raises the current by about 370 mA, so a cut at the 25th percentile of the current plus 150 mA separates the two unambiguously), and the energy is the integral of power above this baseline over the full window, divided by the 20 rounds. Table 7.2 shows the result for both instruments on the same circuit.

Batch size B	Task length (ms)	Energy above baseline (mJ)	Reference (mJ)
179	50	164	168
715	200	415	452
1787	499	1055	1087
3574	999	2017	2119

Table 7.2.: Batch-size experiment on the Raspberry Pi 5 (ResNet-8, LiteRT, $n = 20$ rounds per batch size). Energy above the idle baseline per round; the task length is the device-side median.

Linear fit. For a first model we use $E(B) = E_{\text{fixed}} + B \cdot E_{\text{marginal}}$. It yields $E_{\text{marginal}} = 0.550 \pm 0.006$ mJ per inference on the power meter and 0.577 ± 0.003 mJ on the reference instrument ($R^2 \geq 0.9995$)—the two instruments agree on the slope to within 5%. The intercepts are $E_{\text{fixed}} = 54 \pm 11$ mJ and 55 ± 5 mJ; the two instruments agree on the intercept as well.

The fixed overhead is small and estimator-limited. The intercept is sensitive to the baseline estimate: shifting the baseline by 1 mA moves every round by 15 mJ (5 V over 3 s). With both instruments at 55 ± 11 mJ or better, two standard errors bound the fixed overhead below approximately 0.08 J, but the data does not resolve its exact value. An estimator that instead takes the quietest stretch of each period as the baseline books the idle bursts of the desktop system into every round and raises the intercept to roughly 0.3 J; the range above treats those bursts as part of idle, which is the appropriate convention when the bursts occur with or without the task. On the time side, the cold start adds only 0.26 ms to the first inference, which is negligible in energy.

7.3. Offload Break-Even

The break-even question is: how many inferences per round must be offloaded from the ESP32-S3 to the Pi 5 to save energy?

Let B be the number of inferences per round. On the ESP32-S3, each inference costs 173.2 mJ. On the Pi 5, a round of B inferences costs $E_{\text{fixed}} + 0.55 \cdot B$ mJ, with $E_{\text{fixed}} \leq 0.08$ J (Section 7.2). Even at this upper bound, a single offloaded inference costs less than half of a local one:

$$E_{\text{fixed}} + E_{\text{marginal}} \;\leq\; 80 + 0.6 \text{ mJ} \;<\; 173.2 \text{ mJ} \tag{7.1}$$

Under this baseline convention, and not counting the WiFi transfer of the input data (see the limitations below), offloading saves energy from the first inference of a round, and the advantage grows with the batch size. Table 7.3 lists the measured per-inference energy at each batch size.

Batch size B	Pi 5 per-inference	Ratio to ESP32-S3
179	0.91 mJ	190× cheaper
715	0.58 mJ	298× cheaper
1787	0.59 mJ	293× cheaper
3574	0.56 mJ	307× cheaper

Table 7.3.: Measured per-inference energy on the Pi 5 (energy above baseline per round divided by B) and its ratio to the ESP32-S3's 173.2 mJ per inference.

Interpretation. It can be seen that at every measured batch size, one inference on the Pi 5 costs about 190 to 310 times less energy than on the ESP32-S3. The fixed overhead is too small to change this: its upper bound (0.08 J) stays well below the cost of one local inference, so no minimum batch size is required. What can change the picture is the cost that this comparison excludes—the WiFi transfer of the input data.

Limitations. The largest missing term is communication. Input data still has to be transferred from the ESP32-S3 to the Pi 5, and the WiFi energy depends on payload size, link quality and radio power state. This cost should be measured in the same round rather than taken from a datasheet, and the present platform cannot yet isolate it from the S3 workload. Communication can therefore move the break-even point to the right and, for a small payload, its relative uncertainty may be larger than the measured compute energy itself.

The Pi 5 also ran a full desktop environment. A headless setup would lower its idle baseline and tighten the overhead bound. Finally, the same model used different runtimes (C on the ESP32-S3 and XNNPACK on the Pi 5), so this is a full-stack comparison, not a claim about raw silicon efficiency.

8. Conclusion and Future Work

8.1. Summary

This thesis presents a low-cost, networked power measurement platform for edge devices, covering its design, implementation and evaluation. The platform uses commercial INA current sensors on an ESP32-C6 microcontroller, communicates over WiFi via MQTT and REST, and provides mutually exclusive acquisition modes: averaged monitoring at 1 Hz to 50 Hz, burst and streaming capture at configured rates up to 3000 Hz, and transition capture at a practical 2 kHz (absolute limit: 2.4 kHz). The total parts cost is approximately €27 per three-channel unit.

The questions posed in the introduction concern measurement accuracy, acquisition and transport, and attribution of energy to a task. The paragraphs below summarize the measured answers.

Measurement accuracy. After a two-parameter linear calibration against the Monsoon HVPM, the platform achieves a current residual of $+0.10\%$ on the Raspberry Pi 5 (5 V, 1 A) and -0.06% to -0.14% on the Jetson Orin Nano (12 V, 0.5 A to 1.7 A). The overall energy measurement accuracy is within 3% of the reference instrument across all three DUTs.

Acquisition and transport. The platform provides mutually exclusive acquisition modes in the same firmware. Averaged monitoring delivers continuous data at rates up to 44 Hz at the 50 Hz setting (the deficit varies by session, as discussed in the evaluation) with per-message MQTT transmission in 0.72 ms. Transition capture records individual events at 2 kHz with sub-millisecond boundary placement, triggered by a configurable current threshold. The MQTT transport decouples the device from its consumers: eight concurrent subscribers add zero measurable cost to the device, while the equivalent REST polling at eight consumers produces 5.6% request failures and p95 latency of 1084 ms.

Task energy attribution. A post-hoc alignment method based on cross-correlation of natural load transitions achieves a median residual of 132 µs, with zero hardware

modifications to the DUT and zero clock-synchronization infrastructure. At 2 kHz sampling (500 µs interval), this is smaller than one sample period.

8.2. Future Work

Several directions remain.

Network infrastructure measurement. The current platform mainly measures end devices. Extending the measurement to network switches and routers would capture the communication energy of an offloading round, closing the gap identified in Section 7.3. The INA228's 85 V bus voltage range is sufficient for Power-over-Ethernet switches (48 V).

Improved clock discipline. The firmware uses the ESP-IDF SNTP client (`esp_sntp`), which provides millisecond-level accuracy. Replacing it with `chrony` on a Linux-based measurement host could improve the time base and reduce the alignment residual, particularly for workloads with fewer or weaker load transitions.

Multi-meter clock synchronization. When multiple power meters observe different devices in the same offloading round, their clocks must be synchronized to attribute energy to the same time window. The current platform aligns each meter's data to the host independently. A direct meter-to-meter clock synchronization (e.g., a shared PPS signal or a coordinated MQTT timestamp exchange) would reduce the inter-meter alignment error below the current two-hop accumulation.

Integrated synchronization pipeline. The time synchronization implementation currently runs as a standalone offline script (`marker_align.py`). Integrating it into the orchestrator's task energy pipeline would enable end-to-end energy attribution in a single automated run, without manual post-processing.

Additional DUT coverage. The evaluation covers three devices (ESP32-S3, Raspberry Pi 5, Jetson Orin Nano). Adding devices with different architectures (e.g., RISC-V, FPGA accelerators) and power profiles would test the platform's generality, particularly the transition capture's ability to detect events with different rise times and amplitudes.

A. REST API Reference

For completeness, the power-meter and server endpoints are collected here. Keeping them in an appendix avoids interrupting the implementation discussion with individual request fields. Path parameters are shown in braces. All JSON request and response bodies use `Content-Type: application/json`.

A.1. Device Endpoints

The power meter firmware runs an HTTP server on port 80. Table A.1 lists all endpoints.

Method	Path	Description
GET	`/api/v1/power`	Current monitoring data for all channels (calibrated). Optional query parameter `ch` selects a single channel by index.
POST	`/api/v1/reset`	Reset all energy counters to zero.
GET	`/api/v1/scan`	List of detected sensors (model, address, online status).
GET	`/api/v1/rawscan`	Raw I^2C bus census of addresses `0x08`–`0x77`.
GET	`/api/v1/cal`	Full calibration table (gain, offset per channel; `nvs_ok` flag).
PUT	`/api/v1/cal`	Set calibration for one channel. Body: `{"addr":68,"gain":1.02,"offset_ma":3.1}`.
DELETE	`/api/v1/cal`	Clear calibration for one channel. Query parameter `addr` specifies the I^2C address.
GET	`/api/v1/rate`	Current sampling rate and ADC round time. Returns `hz`, `period_ms`, `adc_round_ms`.
PUT	`/api/v1/rate`	Set sampling rate. Body: `{"hz":20}`. Range: 1 Hz to 50 Hz; persisted in NVS.

Table A.1.: Power meter REST endpoints (port 80).

A.2. Server Endpoints

The server runs on port 8000. Most endpoints are available under both `/api/` and `/api/v1/`; the table shows the canonical path. Table A.2 through Table A.4 group the endpoints by function.

Method	Path	Description
GET	/	Serve the dashboard (single-page application).
GET	/api/history	Down-sampled time series for the last n seconds (default 300, max 86 400). Returns per-channel arrays of voltage, current, power, and energy, plus workload bands.
GET	/api/export.csv	Raw sample rows as CSV for the last n seconds (default 3600).
GET	/api/rate	Last reported sampling rate from each power meter.
PUT	/api/rate	Set the sampling rate on a power meter. Body: {"device":"pma01","hz":20}.
GET	/api/cal	Calibration data: the retained table from each power meter and the least-squares fit records on disk.
POST	/api/reset/{device}	Reset the energy accumulator on the named power meter.

Table A.2.: Server endpoints—monitoring and export.

Method	Path	Description
GET	/api/duts	DUT registry with live state, channel bindings, agent status, and available workloads.
POST	/api/dut/{id}/workload/{wl}	Apply workload wl on DUT id. MQTT DUTs receive the command over MQTT; SSH DUTs receive it over SSH.
POST	/api/dut/{id}/agent/{action}	Start, stop, or query the on-board measurement agent on an SSH DUT. Actions: start, stop, status.
PUT	/api/dut/{id}/channel	Bind or unbind a DUT to a measurement channel. Body: {"channel":"INA228@0x44"} or {"channel":null}.
POST	/api/workload/{cmd}	Publish workload command cmd to the primary DUT (imperative form of PUT /api/v1/workload).
GET	/api/v1/workload	Last reported workload state for the primary DUT.
PUT	/api/v1/workload	Set workload on the primary DUT. Body: {"state":"cpu"}. Allowed values: idle, cpu, mem, wifi_tx, infer, auto, display_on, display_off.

Table A.3.: Server endpoints—DUT and workload control.

Method	Path	Description
POST	`/api/exp/start`	Create an experiment. Body: `{"name":"cal-s3","meta":{...}}`. Returns `exp_id`.
POST	`/api/exp/step/start`	Start a step within an experiment. Body: `{"exp_id":1,"step":"cpu","rep":1}`. Returns `step_id`.
POST	`/api/exp/step/end`	End a step. Body: `{"step_id":5}`.
POST	`/api/exp/end`	End an experiment. Body: `{"exp_id":1}`.
GET	`/api/exp/{id}/summary`	Per-step, per-channel statistics: mean current, standard deviation, mean voltage, mean power, sample count.
GET	`/api/exp/{id}/export.csv`	Raw sample rows for all completed steps of an experiment, as CSV.

Table A.4.: Server endpoints—experiment lifecycle.

Protocol	Path	Description
WS	`/ws`	Live data stream. On connect: snapshot of last monitoring message, calibration table, workload state, and DUT registry. During operation: monitoring data, calibration updates, workload transitions, DUT state changes, and a status heartbeat every 2 s.

Table A.5.: Server WebSocket endpoint.

B. MQTT Topics and JSON Schemas

The MQTT topic tree and the main JSON schemas are given below. They are the wire-level counterpart of the interfaces discussed earlier. In topic patterns, `<id>` is the power meter's device identifier (e.g., `pma01`) and + is the single-level MQTT wildcard.

B.1. Topic Tree

Table B.1 lists all topics grouped by publisher. The *R* column marks retained messages.

Topic	R	Publisher	Purpose
Power meter (device)			
`powermeter/<id>/telemetry`		device	Monitoring message (one per sampling cycle)
`powermeter/<id>/cal`	✓	device	Calibration table (republished on connect and after changes)
`powermeter/<id>/burst`		device	Burst and streaming capture output (binary batches or JSON chunks)
`powermeter/<id>/trigger`		device	Transition capture output (binary batches + JSON summary)
`powermeter/<id>/acc`		device	Energy accumulator readout (INA228 only)
`powermeter/<id>/cmd`		server	Commands: rate change, calibration, capture requests, accumulator, energy reset
DUT workload			
`workload/s3/state`		DUT	Self-reported workload phase
`workload/s3/cmd`		server	Workload commands
Reference instrument			
`monsoon/state`	✓	agent	Monsoon HVPM status (voltage, current, `sim` flag)
`monsoon/cmd`		orchestrator	Monsoon control (`vout`, `safety`)

Table B.1.: MQTT topic tree.

B.2. Message Schemas

This section describes the JSON fields of the three most frequently used message types. The binary batch format used by streaming and transition capture is defined in Table 5.4.

B.2.1. Monitoring Message

Published on `powermeter/<id>/telemetry` once per sampling cycle.

Field	Type	Description
`device_id`	string	Power meter identifier
`ts`	uint32	Unix epoch (seconds)
`ms`	uint32	Device uptime (`millis()`)
`mono_us`	int64	Monotonic clock (µs, `micros()`)
`epoch_us`	int64	Wall clock (µs, SNTP-disciplined; read back to back with `mono_us`)
`smp_ms`	uint32	Current sampling period (ms)
`sched`	0/1	Deadline policy (1 = accumulating)
`disp_ms`	uint32	Display refresh period (0 if display off)
`net_us`	uint32	Network servicing time (µs)
`acq_us`	uint32	Sensor read time (µs)
`pub_us`	uint32	Previous publish time (µs)
`ui_us`	uint32	LCD refresh time (µs)
`channels`	array	Per-channel data (see Table B.3)

Table B.2.: Monitoring message fields, in serialization order.

A representative message with the three sensors of the test bench follows. The device sends it as one line; the listing adds line breaks.

B.2.2. Calibration Table

Published retained on `powermeter/<id>/cal` on every MQTT connect and after any calibration change.

The following table was recorded from the device on 2026-08-19. Four sensors were attached at that time; the second INA228 (at `0x40`) served the dual-chip comparison in Subsection 6.5.3.

Field	Type	Description
`ch`	uint8	Channel index (bus-address order)
`label`	string	Channel identifier (e.g., `"INA228@0x44"`)
`model`	string	Sensor model (`INA219`, `INA226`, or `INA228`)
`addr`	uint8	I^2C address
`online`	bool	Sensor responded on last read
`valid`	bool	Reading passed plausibility checks
`active`	bool	Current above I_{min} (Table 5.1)
`bus_V`	float	Bus voltage (V)
`current_mA`	float	Current (mA, calibrated)
`power_mW`	float	Power (mW, $V \times I$)
`energy_mWh`	float	Cumulative energy (mW h, firmware trapezoidal integral)
`energy_hw_mWh`	float	Cumulative energy (mW h, INA228 hardware accumulator; the field is absent for sensor models without one)

Table B.3.: Per-channel object within a monitoring message.


```
{"device_id":"pma01","ts":1787085512,"ms":5211042,
 "mono_us":5211042451,"epoch_us":1787085512345678,
 "smp_ms":100,"sched":1,"disp_ms":500,
 "net_us":9481,"acq_us":5964,"pub_us":4689,"ui_us":70133,
 "channels":[
  {"ch":0,"label":"INA219@0x41","model":"INA219","addr":65,
   "online":true,"valid":true,"active":true,
   "bus_V":4.952,"current_mA":121.30,"power_mW":600.68,
   "energy_mWh":152.3012},
  {"ch":1,"label":"INA228@0x44","model":"INA228","addr":68,
   "online":true,"valid":true,"active":true,
   "bus_V":19.132,"current_mA":343.58,"power_mW":6573.37,
   "energy_mWh":505.9014,"energy_hw_mWh":517.1352},
  {"ch":2,"label":"INA226@0x45","model":"INA226","addr":69,
   "online":true,"valid":true,"active":true,
   "bus_V":5.023,"current_mA":777.09,"power_mW":3903.32,
   "energy_mWh":1203.4406}]}
```

Field	Type	Description
`channels`	array	One object per channel
`nvs_ok`	bool	NVS persistence healthy
`ts`	uint32	Unix epoch (seconds)
Per-channel object		
`addr`	uint8	I^2C address
`label`	string	Channel identifier
`gain`	float	Gain factor (default 1.0)
`offset_ma`	float	Offset (mA, default 0.0)
`attached`	bool	Sensor currently detected on the bus

Table B.4.: Calibration table fields.

```
{"channels":[
  {"addr":64,"label":"INA228@0x40","gain":1.013054,
   "offset_ma":3.826,"attached":true},
  {"addr":65,"label":"INA219@0x41","gain":0.997887,
   "offset_ma":4.136,"attached":true},
  {"addr":68,"label":"INA228@0x44","gain":1.013195,
   "offset_ma":4.469,"attached":true},
  {"addr":69,"label":"INA226@0x45","gain":1.004140,
   "offset_ma":0.790,"attached":true}],
 "nvs_ok":true,"ts":1787095512}
```

B.2.3. Command Messages

Published on `powermeter/<id>/cmd` by the server or analysis scripts. The plain-text string `"reset"` resets all energy counters. All other commands are JSON objects; the top-level key selects the command.

Command key	Payload and effect
`rate`	`{"rate":{"hz":20}}` — Set the sampling rate (1 Hz to 50 Hz).
`set_cal`	`{"set_cal":{"addr":68,"gain":1.02, "offset_ma":3.1}}` — Write calibration for one channel (`label` may replace `addr`).
`clear_cal`	`{"clear_cal":{"addr":68}}` — Reset one channel to identity (1.0 / 0.0).
`get_cal`	`{"get_cal":true}` — Republish the calibration table.
`burst`	`{"burst":{"addr":68,"hz":500,"seconds":4}}` — Start a burst capture. Rate 50 Hz to 3000 Hz; duration 1 s to 10 s.
`stream`	`{"stream":{"addr":68,"hz":2000,"seconds":60}}` — Start a streaming capture. Rate 100 Hz to 3000 Hz; duration up to 120 s.
`trigger`	`{"trigger":{"addr":68,"level_ma":900, "sustain_ms":5, "pre_ms":50,"post_ms":300, "hz":2000,"timeout_s":30}}` — Arm a transition capture. Rate 100 Hz to 3000 Hz; timeout up to 1800 s.
`acc`	`{"acc":{"addr":68}}` — Read the INA228 energy accumulator. Add `"reset":1` to reset before reading.

Table B.5.: Command message formats.

C. Bench Photographs

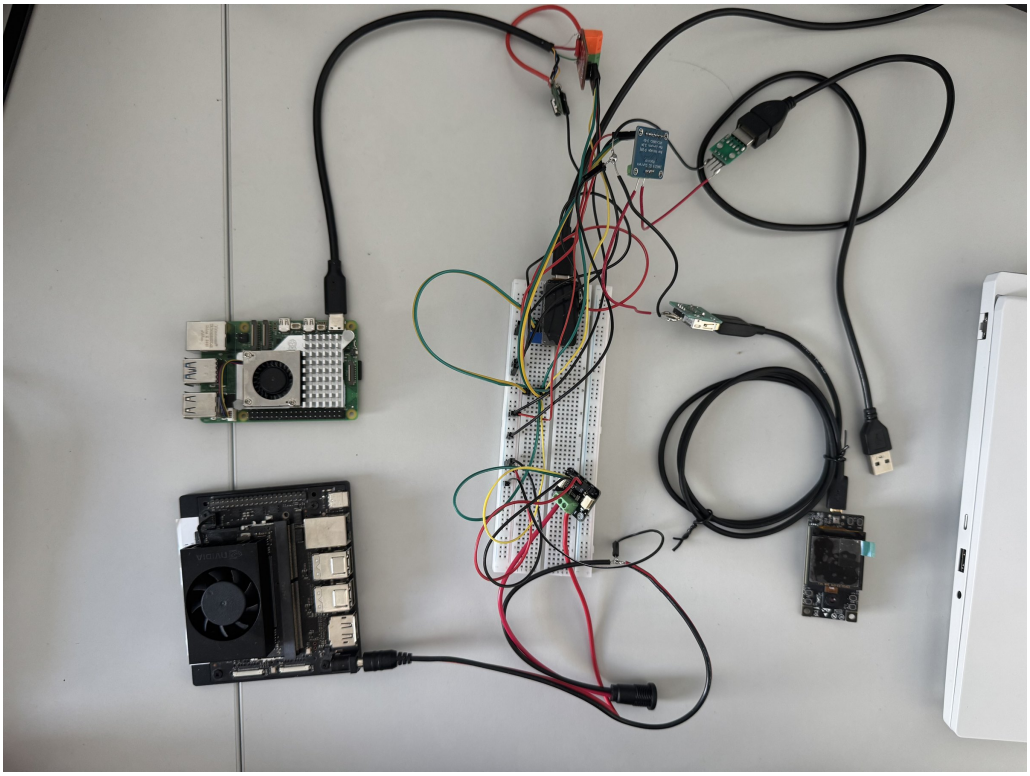


Figure C.1.: The measurement bench. Raspberry Pi 5 and Jetson Orin Nano on the left, the sensor modules on the breadboard in the center, and the ESP32-S3 DUT on the right. Each DUT's supply cable is opened at an adapter and routed through its sensor module.

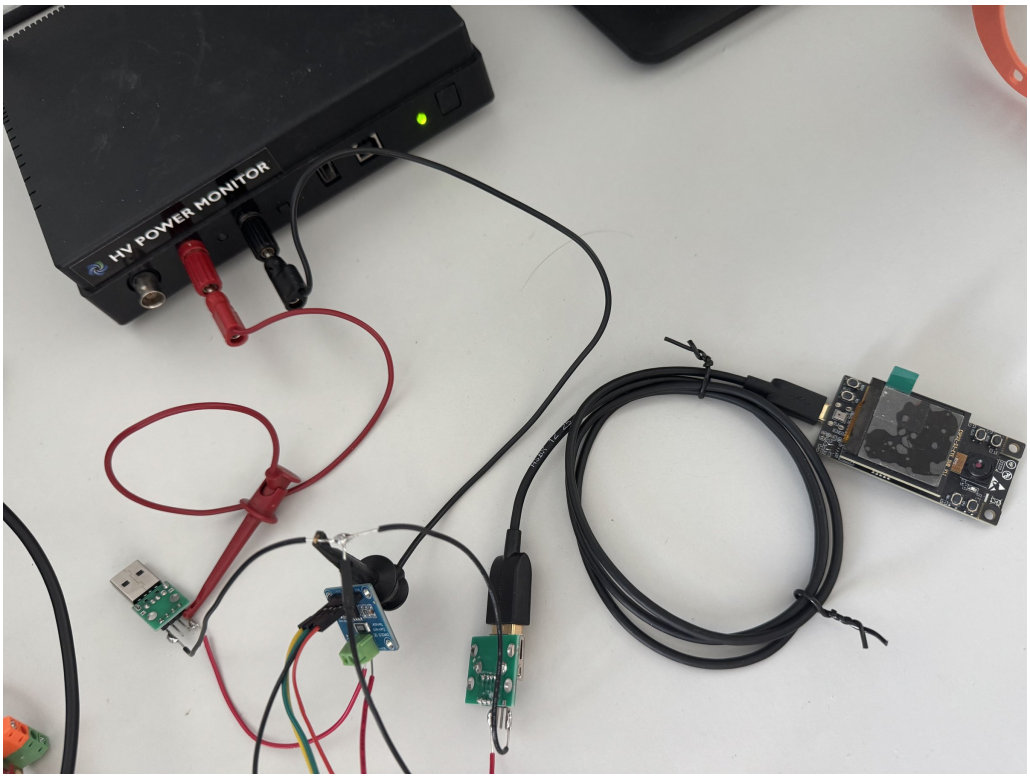


Figure C.2.: Calibration setup on the ESP32-S3 supply, as wired in Figure 6.1. The reference instrument (Monsoon HVPM, top left) provides the 5 V input; its output passes through the INA219 sensor module (center) to the ESP32-S3 DUT (right).

Abbreviations

ADC analog-to-digital converter

CPU central processing unit

DUT device under test

GPIO general-purpose input/output

GPU graphics processing unit

HTTP Hypertext Transfer Protocol

HVPM High Voltage Power Monitor

IoT Internet of Things

JSON JavaScript Object Notation

MQTT Message Queuing Telemetry Transport

NTP Network Time Protocol

NVS non-volatile storage

PMIC power-management integrated circuit

REST Representational State Transfer

SNTP Simple Network Time Protocol

USB Universal Serial Bus

List of Figures

List of Tables

Bibliography

[1] C. Banbury, V. J. Reddi, P. Torelli, N. Jeffries, C. Kiraly, J. Holleman, P. Montino, D. Kanter, P. Warden, D. Pau, U. Thakker, A. Torrini, J. Cordaro, G. Di Guglielmo, J. Duarte, H. Tran, N. Tran, W. Niu, and X. Xu. "MLPerf Tiny Benchmark." In: *Proc. NeurIPS Datasets and Benchmarks Track*. 2021.

[2] L. Bantel, S. Rottacker, and D. Pflüger. *An Open-Source Power Measurement Platform for System-Level Semiconductor Testing*. arXiv:2608.05888. 2026.

[3] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. "sMAP: A Simple Measurement and Actuation Profile for Physical Information." In: *Proc. ACM SenSys*. 2010. DOI: 10.1145/1869983.1870003.

[4] B. Dezfouli, I. Amirtharaj, and C.-C. Li. "EMPIOT: An Energy Measurement Platform for Wireless IoT Devices." In: *Journal of Network and Computer Applications* 121 (2018), pp. 135–148. DOI: 10.1016/j.jnca.2018.07.016.

[5] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. "The Many Faces of Publish/Subscribe." In: *ACM Computing Surveys* 35.2 (2003), pp. 114–131. DOI: 10.1145/857076.857078.

[6] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. "Quanto: Tracking Energy in Networked Embedded Systems." In: *Proc. USENIX OSDI*. 2008.

[7] K. Geissdoerfer, M. Chwalisz, and M. Zimmerling. "Shepherd: A Portable Testbed for the Batteryless IoT." In: *Proc. ACM SenSys*. 2019. DOI: 10.1145/3356250.3360042.

[8] K. Geissdoerfer, I. Splitt, M. Sokolowski, C. Herrmann, J. Kubicki, J. de Winkel, and M. Zimmerling. "Shepherd Nova: A Public Testbed for Rigorous Experiments Under Repeatable Energy-Harvesting Conditions." In: *Proc. ACM MobiSys*. 2025, pp. 236–248. DOI: 10.1145/3711875.3729146.

[9] W. Geng, O. K. Altas, D. Guzman, G. Bartolomeo, N. Mohan, and J. Ott. "KUT: Towards Lightweight On-path Network Assessment for Edge Orchestration." In: *Proc. ACM CoNEXT*. Poster. 2025.

[10] W. Geng, N. Mohan, and J. Ott. "Budget-Adaptive Routing: Skipping the Weak When the Strong Answers Anyway." In: *Proc. ACM SIGCOMM Workshop on Networks for AI Computing (NAIC)*. 2026.

[11] W. Geng, X. Su, N. Mohan, J. Ott, and P. Hui. "SMOOTH: Scalable Multitask Offloading with Backbone Sharing." In: *Proc. IFIP Networking*. 2026.

[12] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat. "Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization." In: *Proc. USENIX NSDI*. 2018, pp. 81–94.

[13] O. Gnawali, K.-Y. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler. "The Tenet Architecture for Tiered Sensor Networks." In: *Proc. ACM SenSys*. 2006, pp. 153–166. DOI: 10.1145/1182807.1182823.

[14] J. W. Hui and D. E. Culler. "IP is Dead, Long Live IP for Wireless Sensor Networks." In: *Proc. ACM SenSys*. 2008. DOI: 10.1145/1460412.1460415.

[15] Y. Li, G. Kumar, H. Hariharan, H. M. G. Wassel, P. Hochschild, D. Platt, S. L. Sabato, M. Yu, N. Dukkipati, P. Chandra, and A. Vahdat. "Sundial: Fault-tolerant Clock Synchronization for Datacenters." In: *Proc. USENIX OSDI*. 2020, pp. 1171–1186.

[16] Z. Lu. "Test-Time Accuracy Indicators for Object Detection." Bachelor's Thesis. Technical University of Munich, 2026.

[17] D. L. Mills. *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space*. 2nd ed. CRC Press, 2010.

[18] Monsoon Solutions Inc. *High Voltage Power Monitor (HVPM)*. 2020.

[19] OASIS. *MQTT Version 5.0*. OASIS Standard. 2019.

[20] A. Schulman, T. Thapliyal, S. Katti, N. Spring, D. Levin, and P. Dutta. *BattOr: Plug-and-debug Energy Debugging for Applications on Smartphones and Laptops*. Tech. rep. Stanford University, 2016.

[21] N. Shalavi, A. Khoshsirat, M. Stellini, A. Zanella, and M. Rossi. "Accurate Calibration of Power Measurements from Internal Power Sensors on NVIDIA Jetson Devices." In: *Proc. IEEE EDGE*. 2023, pp. 166–170. DOI: 10.1109/EDGE60047.2023.00034.

[22] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. "Edge Computing: Vision and Challenges." In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[23] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele. "Measurement and Validation of Energy Harvesting IoT Devices." In: *Proc. Design, Automation & Test in Europe (DATE)*. 2017, pp. 1159–1164. DOI: 10.23919/DATE.2017.7927164.

[24] Texas Instruments. *INA219 Bidirectional Current/Power Monitor*. SBOS448G, Rev. G. 2015.

[25] Texas Instruments. *INA226 High-Side or Low-Side Measurement, Bi-Directional Current and Power Monitor*. SBOS547A. 2015.

[26] Texas Instruments. *INA228 85-V, 20-Bit, Ultra-Precise Power/Energy/Charge Monitor*. SLYS021A. 2021.

[27] R. Trüb, R. Da Forno, L. Sigrist, L. Mühlebach, A. Biri, J. Beutel, and L. Thiele. "FlockLab 2: Multi-Modal Testing and Validation for Wireless IoT." In: *Proc. Workshop on Benchmarking Cyber-Physical Systems and Internet of Things (CPS-IoTBench)*. 2020.

[28] A. Tschand, A. T. R. Rajan, S. Idgunji, A. Ghosh, J. Holleman, C. Kiraly, P. Ambalkar, R. Borkar, R. Chukka, T. Cockrell, O. Curtis, G. Fursin, M. Hodak, H. Kassa, A. Lokhmotov, D. Miskovic, Y. Pan, M. P. Manmathan, L. Raymond, T. St. John, A. Suresh, R. Taubitz, S. Zhan, S. Wasson, D. Kanter, and V. J. Reddi. "MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from Microwatts to Megawatts for Sustainable AI." In: *Proc. IEEE HPCA*. 2025, pp. 1201–1216. DOI: 10.1109/HPCA61900.2025.00092.

[29] S. van der Vlugt, L. Oostrum, G. Schoonderbeek, B. van Werkhoven, B. Veenboer, K. Doekemeijer, and J. W. Romein. "PowerSensor3: A Fast and Accurate Open Source Power Measurement Tool." In: *Proc. IEEE ISPASS*. 2025, pp. 213–226. DOI: 10.1109/ISPASS64960.2025.00028.

[30] Z. Yang, K. Adámek, and W. Armour. "Accurate and Convenient Energy Measurements for GPUs: A Detailed Study of NVIDIA GPU's Built-In Power Sensor." In: *Proc. IEEE/ACM SC*. 2024. DOI: 10.1109/SC41406.2024.00028.