DISTRIBUTED STREAM PROCESSING

Edge to Cloud Stream Processing for Autonomous Vehicle Perception Logs

Pratik Mulik

Department of AI&ML, Navsahyadri Group of Institutions, Pune, Maharashtra, India, Mentor: Prof. Archana Pawar, Department of AI&ML, Navsahyadri Group of Institutions, Pune, Maharashtra, India

Published November 2026

Abstract

Small shops, laboratories and farms often need nothing more than a way to look at an area from another room, yet a conventional CCTV set-up asks for cameras, cables, a recorder and a monitor. This paper describes Sentinel Vision, a surveillance camera built around one ESP32-CAM board. The board joins the local Wi-Fi network and runs two small web servers: one on port 80 that delivers a dashboard showing camera status, uptime and Wi-Fi signal strength, and one on port 81 that streams live motion-JPEG video to any web browser. Because the page and the video are served separately, the dashboard stays responsive while the stream is running. We explain the hardware, the firmware structure, the camera settings that adapt to the memory available on the board, and the browser logic that controls the stream. We also say plainly what the prototype does not do: it has no login or encryption, serves one viewer at a time, and performs no detection, recording or alerting. What it offers is a cheap, easily reproduced base for local monitoring and for more intelligent extensions later.

Keywords: embedded systems; ESP32-CAM; Internet of Things; MJPEG streaming; video surveillance

1 Introduction

Watching over a place used to mean standing in it. Today a camera does the standing for us, and that has quietly become part of ordinary life: a shopkeeper checks the store room from the counter, a laboratory in-charge glances at the entrance from the staff room, a farmer looks at a shed from the house. The idea is simple. The equipment that usually delivers it is not. A traditional CCTV installation needs cameras, power and video cabling, a digital or network video recorder and a monitor to view the result, and the video can normally be seen only where the recorder sits [1].

Over the last few years small Wi-Fi microcontrollers with a camera attached have changed what is possible. The ESP32-CAM is a good example. It combines a dual-core ESP32 processor, a wireless radio and an OV2640 two-megapixel sensor on a board not much bigger than a postage stamp [8], [11]. It can capture JPEG frames and send them over a network without any help from another computer.

Still, a board that is merely able to stream is not yet a usable camera. Someone has to decide how the viewing page should look, what should happen when the Wi-Fi connection fails, how to keep the page alive while video is flowing, and how a user can tell whether the camera is healthy or has silently stopped. Sentinel Vision is our attempt to answer these practical questions with a single Arduino sketch and nothing else on the hardware side.

The contributions of this work are modest but concrete:

- a single-board design in which the ESP32-CAM hosts both a viewing dashboard and a live video stream, so that a phone or laptop with a browser is all the viewer needs;
- a firmware structure that serves the dashboard on port 80 and the stream on port 81, with the video handled in a separate FreeRTOS task [13];
- a dark-themed browser dashboard that reports camera status, uptime and Wi-Fi signal, and that exposes explicit stream states to the user; and
- an honest account of the limits of the prototype, together with a path for removing each of them.

The rest of the paper is organised as follows. The next section reviews related approaches. The design and the implementation are then presented, followed by how the prototype is checked and what we plan to measure. The paper then discusses limitations and future work and ends with a conclusion.

2 Related work and background

2.1 Conventional and commercial cameras

Wired CCTV is dependable and records continuously, which is why it remains the default for large premises. Its drawbacks are cost and installation effort, and its video stays tied to the recorder unless more equipment is added. Commercial network and cloud cameras are far easier to set up and can be watched from anywhere through a mobile application, but they are closed products: the user cannot see or change how they work, and some depend on a continuing online service. For a student project, or for a small user who wants a camera that can be understood and modified, neither option is a perfect fit. Table 1 summarises the comparison.

Table 1. Comparison of surveillance approaches.

Approach	Strength	Weakness
Wired CCTV	Reliable; records continuously	Costly; hard to install
Commercial cloud camera	Easy app; remote viewing	Closed design; may need a subscription
Sentinel Vision	Low cost; open; any browser	Local network only; no login yet

2.2 Embedded camera platforms

Espressif publishes a camera driver for the ESP32 family [10] and ships a CameraWebServer example with its Arduino core [9]. That example already does something clever: it moves the video stream onto its own port so that the control page and the stream do not compete. Many hobby and classroom projects start from it. We build on the same idea, but we reorganised the code around a small purpose-written dashboard, a status service and an explicit stream task, so that the whole system is easy to read, explain and extend.

2.3 Streaming video over the web

There are several ways to deliver live video. RTSP [5] is common in professional cameras but needs a player that understands it. HTTP Live Streaming [6] works in browsers, but it cuts video into segments and therefore adds delay, and it needs an encoder that a small microcontroller does not have. Motion-JPEG over HTTP is the humble alternative. Each frame is an independent JPEG image [4], sent as one part of a multipart response [3] that the browser keeps open. It uses more bandwidth than a

video codec, but it needs no encoder (the OV2640 produces JPEG itself), it adds very little delay, and it can be shown with an ordinary image element. For a board with limited processing power these properties matter more than efficiency.

3 System design

3.1 Design goals

Before writing any code we agreed on five goals. First, one board and one sketch: no extra sensors, relays or modules. Second, nothing to install on the viewing device. Third, a page that stays responsive even while video is running. Fourth, honest status: the user should be able to see at a glance whether the camera is alive and how strong its Wi-Fi link is. Fifth, fail loudly: if the camera or the network cannot start, the board should say so on the serial port rather than sit there looking healthy.

3.2 Hardware and overall architecture

The only hardware is an AI-Thinker ESP32-CAM, powered from a 5 V supply. A USB-to-serial adapter is needed once, to upload the sketch, and can be removed afterwards. The overall arrangement is shown in Figure 1. The OV2640 sensor delivers JPEG frames to the ESP32, which exposes them through two services. The dashboard service listens on port 80, and the stream service listens on port 81. Both are reached through the local router or phone hotspot by a browser running on a phone, tablet or laptop.

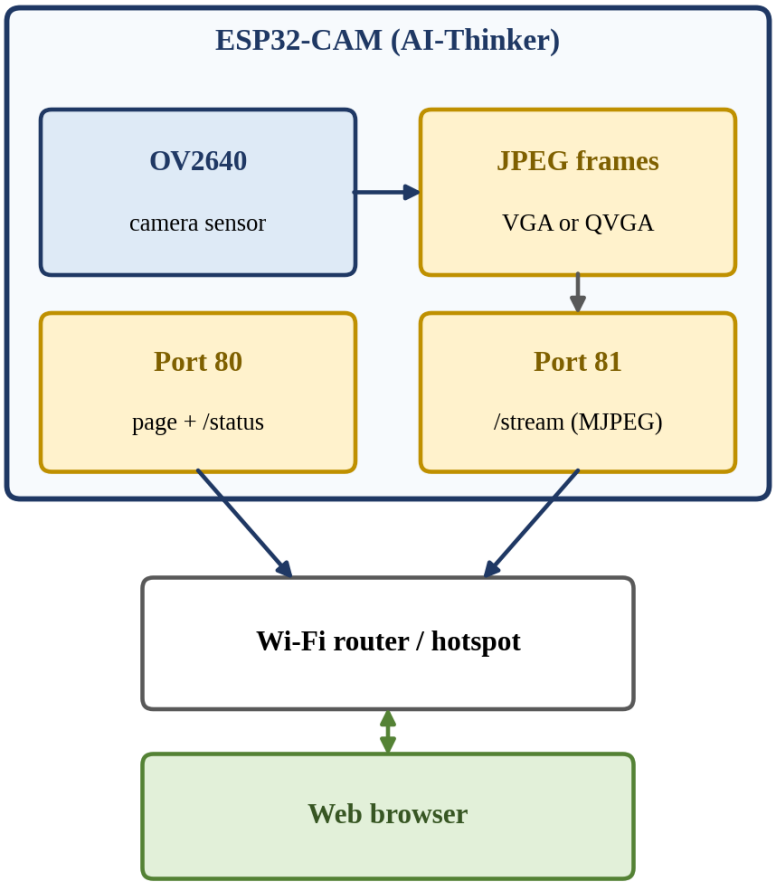


Figure 1. Architecture of Sentinel Vision. One board hosts the dashboard (port 80) and the stream (port 81).

3.3 Software organisation

The sketch is written for the Arduino IDE [12] and is divided into clearly separated parts: Wi-Fi credentials, camera pin definitions, the dashboard page stored as a block of text, a camera set-up function, the dashboard routes, the stream task, and the usual setup() and loop() functions. Keeping these parts apart made the code easier to debug, because a failure could be traced to one stage: camera, network, page or stream.

3.4 The dashboard

The dashboard is a single web page held in program memory (Figure 2). It has a dark theme that suits a monitoring screen, a header with a connection indicator (CONNECTING, ONLINE or OFFLINE), three cards for camera status, device uptime and Wi-Fi signal, a panel for the live feed, and four buttons: Start stream, Stop stream, Refresh status and Reload dashboard. We deliberately kept the page free of external libraries and images, so that it loads even on a network with no internet access.

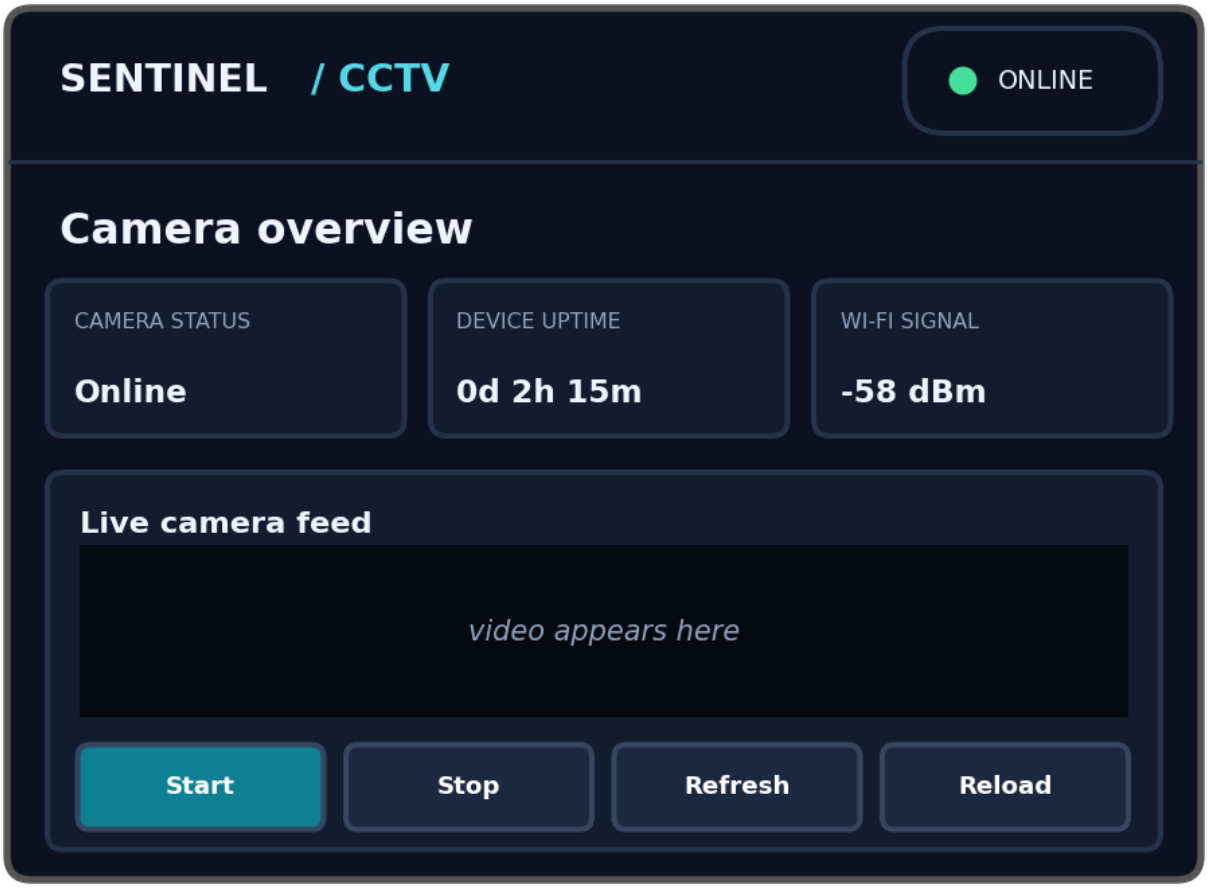


Figure 2. Layout of the Sentinel / CCTV dashboard (values shown are examples).

3.5 Design decisions and trade-offs

Three decisions deserve a short explanation. The first is the choice of motion-JPEG instead of a modern video codec. A codec would cut the bandwidth, but it would force the ESP32 to encode every frame itself, which is hard to do at a useful frame rate on this board. The camera already produces JPEG, so we simply pass those frames on. The second is the separation of the dashboard and the stream. With the simple WebServer library, a long-running stream would hold the server and block every other request, and the status cards would stop updating just when the user wants to look at

them. The third is leaving authentication out of this first version. That was not an oversight: adding a login to a streaming address affects the browser, the server and the way the image is loaded, and we preferred to get the basic path right and well understood before building on it.

4 Implementation

4.1 Camera initialisation

The sketch uses the standard pin map of the AI-Thinker board: eight data lines, the camera clock, pixel clock, VSYNC and HREF signals, and a two-wire configuration bus. The camera is set to deliver JPEG directly, with a 20 MHz clock. At start-up the code asks whether PSRAM is present and picks its settings accordingly (Table 2). With PSRAM it uses VGA frames and two frame buffers in the grab mode that always returns the latest frame; without PSRAM it falls back to a smaller QVGA frame and one buffer. The reasoning is practical. A slow viewer should see the newest picture, not a queue of old ones, and a board without PSRAM should still work rather than refuse to start.

Table 2. Camera settings chosen at start-up.

Setting	With PSRAM	Without
Frame size	VGA 640×480	QVGA 320×240
JPEG quality value	12	15
Frame buffers	2	1
Buffer location	PSRAM	DRAM
Grab mode	Latest frame	When empty

4.2 Wi-Fi connection

The ESP32 is placed in station mode, Wi-Fi power saving is switched off to avoid bursts of delay in the video, and the board waits up to 30 seconds for the network. If the connection is not made, the sketch prints a message and stops. We preferred a clear failure that points to wrong credentials over a half-working device that leaves the user guessing. When the connection succeeds, the serial monitor prints the line CCTV READY followed by the dashboard and stream addresses.

4.3 Dashboard and status service

The dashboard server uses the WebServer library on port 80 and offers two services, listed in Table 3. The root address returns the page, which is sent straight from program memory so that it does not use up working RAM. The /status address returns a very small JSON message with the uptime in seconds and the Wi-Fi signal strength (RSSI) in dBm. Anything else receives a 404 reply.

Table 3. Network services of the camera.

Service	Request	Port	Reply
Dashboard	GET /	80	HTML page
Status	GET /status	80	JSON: uptime, rssi
Live video	GET /stream	81	Multipart JPEG

4.4 Streaming task

The video is served by a plain WiFiServer on port 81, handled in its own FreeRTOS task with an 8 KB stack, pinned to core 1. The task waits for a browser, reads the request line, discards the remaining headers and checks that the request is for /stream. Any other request is answered with 404. For a valid request it sends a header announcing a multipart response with the boundary “frame”, and then loops: capture a frame, write a part header with the frame length, write the JPEG bytes, return the buffer and pause for 30 ms. The pause is small but important, because it gives the rest of the system time to run, including the dashboard. The loop ends when the browser disconnects, the Wi-Fi link drops or a frame cannot be captured (Figure 3).

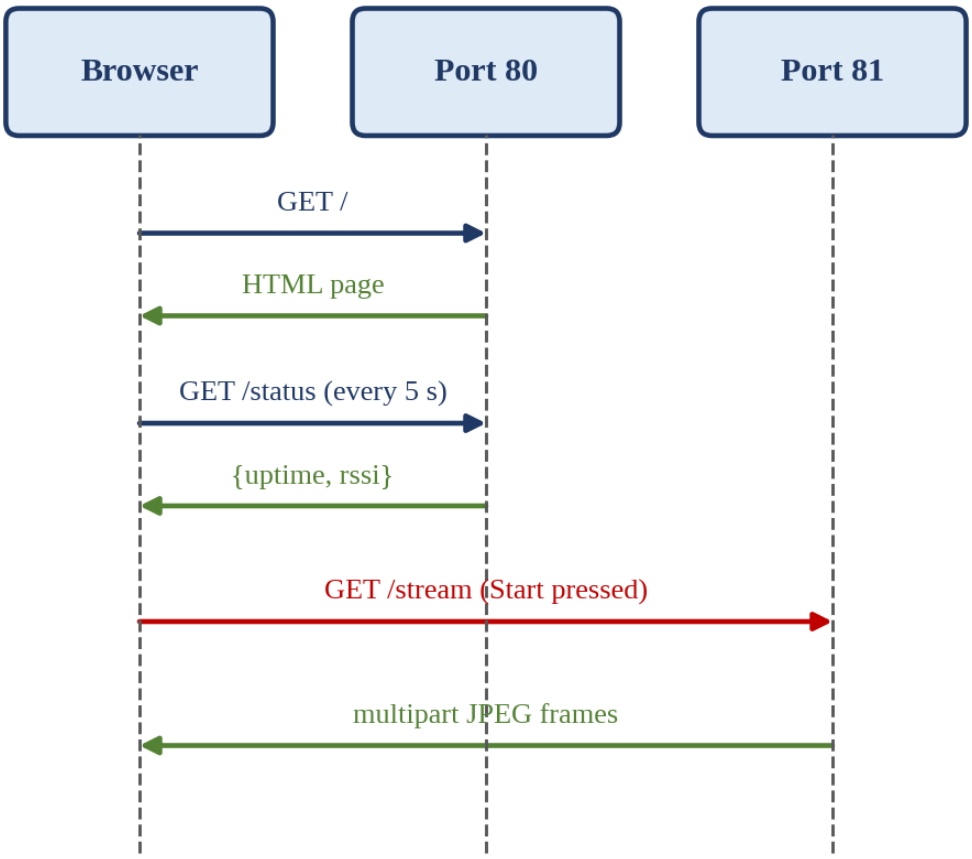


Figure 3. Request sequence between the browser and the two services of the camera.

4.5 Browser logic

A short JavaScript program in the page does three jobs. It polls /status every five seconds, with caching disabled, and converts the uptime into days, hours and minutes. It switches the connection indicator between ONLINE and OFFLINE according to whether the poll succeeds. And it controls the stream. Pressing Start sets the source of an image element to the stream address, with a time stamp added so that the browser does not reuse an old connection; the text above the feed then reads STREAM REQUESTED, and becomes STREAM ACTIVE when the first frame loads. If the image reports an error the text changes to RECONNECT REQUIRED, and pressing Stop clears the image and returns to STREAM STOPPED (Figure 4).

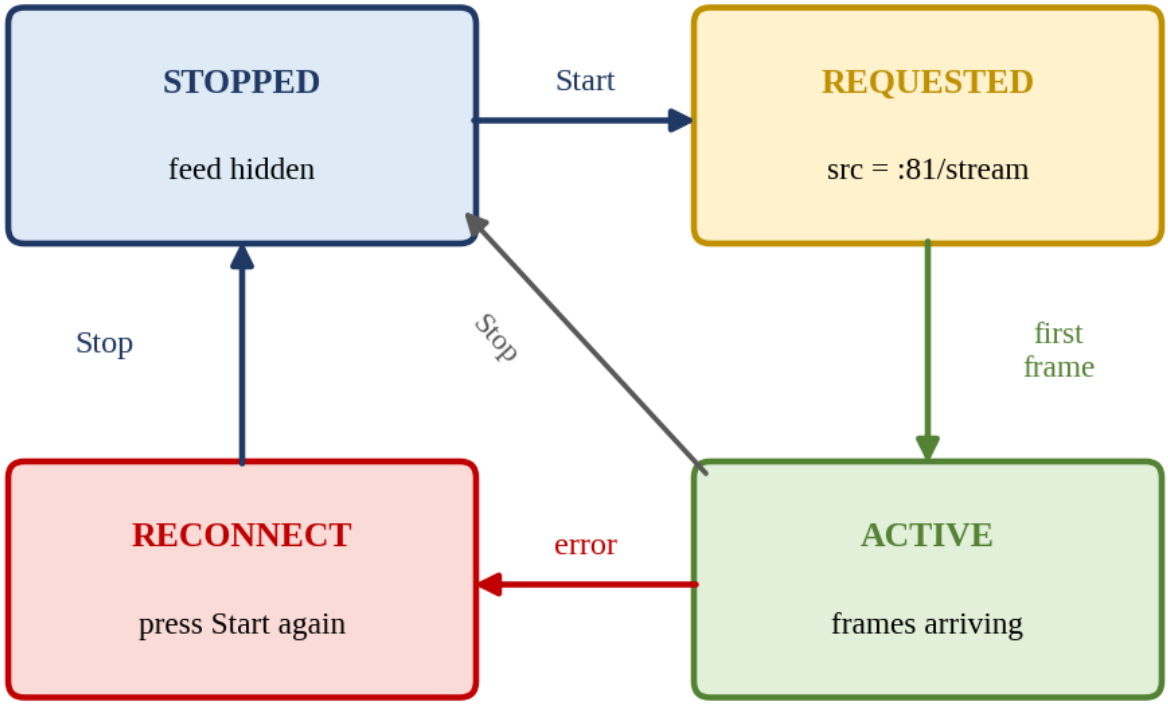


Figure 4. States of the live stream as shown on the dashboard.

4.6 Robustness

A few small checks keep the prototype predictable. The camera is started before Wi-Fi, so a seating or wiring fault is reported at once and is not mistaken for a network problem. The stream loop tests both the client connection and the Wi-Fi status, and ends cleanly when either is lost. Each captured frame buffer is handed back to the camera driver even when a write fails, which stops the driver from running out of buffers. On the browser side, the status request is sent with caching disabled so that an old answer is never mistaken for a live one, and an error from the image is shown as RECONNECT REQUIRED instead of leaving a frozen picture on the screen.

5 Testing approach and discussion

5.1 Functional checks

The prototype is checked function by function, from the board outwards (Table 4). Each check has a clear, observable result, which made it easy to see where a problem lay when something did not work.

Table 4. Functional checks and expected outcome.

Check	Expected outcome
Upload and reset	Serial monitor shows CCTV READY and the IP address
Open dashboard address	Camera overview page appears
Wait for status refresh	Status Online; uptime and signal shown
Press Start stream	Live feed appears; STREAM ACTIVE

Check	Expected outcome
Press Stop stream	Feed hides; STREAM STOPPED
Switch camera off	Dashboard shows OFFLINE

5.2 What we plan to measure

We want to be careful here. This paper reports the design and implementation of the prototype, and it does not claim numerical performance, because systematic measurements have not yet been completed. For the next phase we have chosen four measures. Frame rate is the number of frames received divided by the time taken. Start-up time runs from power-on to the CCTV READY message. Streaming delay can be found by pointing the camera at a running stopwatch shown on a screen placed beside the browser window, photographing both, and reading the difference between the two times. The fourth measure is the RSSI value already displayed on the dashboard, which can be recorded at different distances from the router.

5.3 Factors we expect to matter

Several design choices influence what the viewer sees. Larger frames and a lower JPEG quality number give a sharper picture, but each frame is bigger, so the frame rate falls, especially on a weak signal. Poor lighting makes the sensor expose for longer, which also slows the stream and adds noise. The board warms up during long streaming sessions, so ventilation is advisable. Finally, because the stream task serves one connection at a time, a second viewer has to wait until the first has disconnected. None of these is surprising, but each is worth stating because it decides where the prototype can sensibly be used.

5.4 Where the prototype fits

The prototype suits places where a trusted Wi-Fi network already exists and where a live view, rather than a recording, is what matters. A shop owner can point the board at the entrance and watch it from the counter; a laboratory in-charge can look at a store room from the staff room. A farm shed is a possible fit only where the signal reaches and a steady 5 V supply is available. In every case the camera should face away from windows and strong lights, because a small sensor copes poorly with strong back-light.

6 Limitations and security considerations

The most important limitation is security. The prototype has no login and no encryption, and its stream response even carries a header that allows any web origin to read it, which is convenient while developing but not something to leave in a real deployment. It must therefore be used only on a trusted private network, and its ports must never be exposed directly to the internet. Anyone who can reach the address can watch the video.

There are functional limits too. The present version shows live video and nothing more: it does not detect motion or people, does not record, and does not send alerts. The OV2640 has no night vision. The picture is limited to VGA or QVGA. And a camera that watches people raises a responsible-use question that no firmware can answer: it should be placed only where monitoring is lawful, the people concerned should know it is there, and the video should be visible only to those entitled to see it. In this design the video is not stored anywhere, which at least limits the risk.

7 Future work

The limits above point directly to the next steps. A password-protected login and secure access, for example through a VPN or a reverse proxy, should come first. After that, the stream server can be redesigned to serve several viewers at once, and resolution and quality can be made adjustable from the dashboard. On the functional side we would like to add motion detection, recording to a memory card, and alerts to the owner's phone, and to experiment with infrared lighting for dark areas. Several cameras, powered by batteries or solar panels, could eventually share one dashboard.

8 Conclusion

Sentinel Vision shows that one inexpensive ESP32-CAM board is enough to build a working, understandable surveillance camera that any browser on the local network can watch. Separating the dashboard from the stream, adapting the camera settings to the memory available, and being explicit about stream states and start-up failures turned a bare video feed into something a person can actually use. The prototype is deliberately modest, and its limits are stated openly, but they are the kind that can be removed one at a time. We hope the design is useful as a base for students and small users who want to build on it.

Appendix

Table 5 lists the main parameters used in the sketch, for readers who wish to reproduce or modify the prototype.

Table 5. Key firmware parameters.

Parameter	Value
Dashboard port	80
Stream port	81
Wi-Fi wait time	30 s
Status refresh	5 s
Pause after each frame	30 ms
Stream task stack	8192 bytes
Camera clock	20 MHz
Serial speed	115200 baud

Acknowledgment

The authors would like to thank their mentor, Prof. Archana Pawar, for her patient guidance and encouragement throughout the work, and the Department of AI&ML, Navsahyadri Group of Institutions, Pune, for providing the facilities needed to carry it out.

References

[1] Fielding RT. *Architectural styles and the design of network-based software architectures* [PhD dissertation]. Irvine, CA: University of California; 2000.

[2] Fielding R, Reschke J. *Hypertext Transfer Protocol (HTTP/1.1): message syntax and routing*. RFC 7230. IETF; 2014 Jun.

[3] Freed N, Borenstein N. *Multipurpose Internet Mail Extensions (MIME) part two: media types*. RFC 2046. IETF; 1996 Nov.

[4] International Telecommunication Union. *Digital compression and coding of continuous-tone still images – Requirements and guidelines*. ITU-T Rec. T.81; 1992.

[5] Schulzrinne H, Rao A, Lanphier R. *Real Time Streaming Protocol (RTSP)*. RFC 2326. IETF; 1998 Apr.

[6] Pantos R, May W. *HTTP Live Streaming*. RFC 8216. IETF; 2017 Aug.

[7] IEEE. *IEEE standard for information technology – Telecommunications and information exchange between systems – Part 11: Wireless LAN MAC and PHY specifications*. IEEE Std 802.11.

[8] Espressif Systems. *ESP32 series datasheet*. Shanghai, China [Internet]. Available from: https://espressif.com.

[9] Espressif Systems. *Arduino core for the ESP32 – documentation and CameraWebServer example* [Internet]. Available from: https://docs.espressif.com.

[10] Espressif Systems. *esp32-camera: ESP32 camera driver* [Internet]. Available from: https://github.com/espressif/esp32-camera.

[11] OmniVision Technologies. *OV2640 color CMOS UXGA (2.0 megapixel) CameraChip datasheet*.

[12] Arduino. *Arduino IDE documentation* [Internet]. Available from: https://docs.arduino.cc.

[13] FreeRTOS. *FreeRTOS kernel reference manual* [Internet]. Available from: https://freertos.org.

[14] Mozilla Developer Network. *HTTP overview and multipart content types* [Internet]. Available from: https://developer.mozilla.org.