RECOGNIZE THE MATERIALS

Meet the sender and receiver

01

Send a training Wi-Fi configuration by radio, read what each checkpoint proves, and keep a USB way back.



ESP32-S3-N16R8 board

Runs the receiver firmware.



USB data cable

Flashes firmware and carries console messages.



Computer

Builds the app; watches the USB console.



Android phone

Runs the classic EspTouch sender.



Training Wi-Fi router

Provides a visible 2.4 GHz WPA2-Personal network.

Before you start

Know how to identify the current serial port and save your MicroPython project files. Use a computer with Python 3 for the offline exercise.

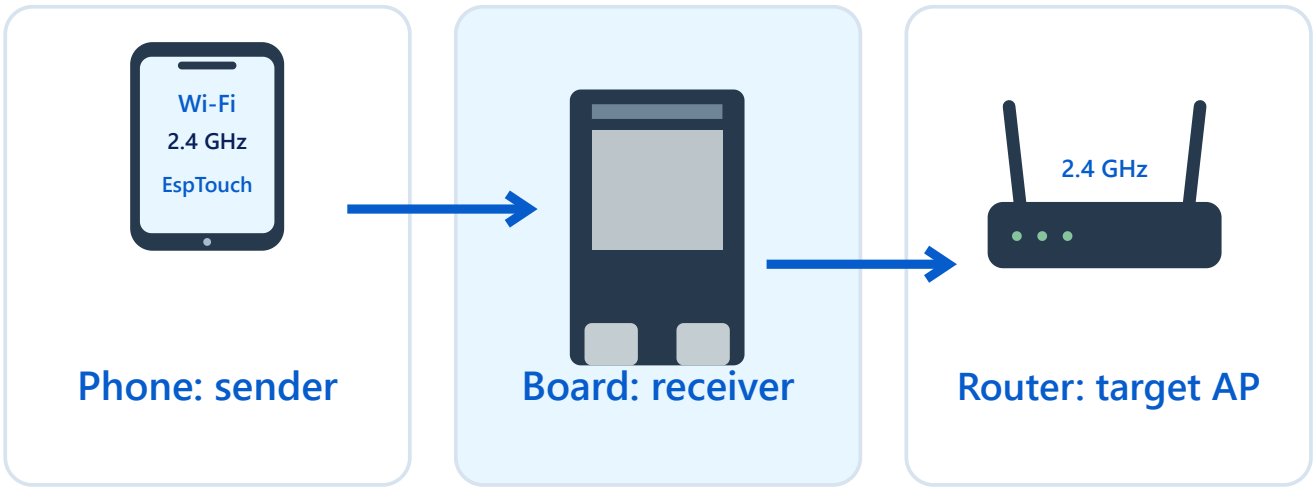
The live exercise uses the supplied ESP-IDF receiver project and Espressif's Android sender. If you cannot install these yet, complete the offline status exercise on page 9 first.

Training network only. Use isolated, non-sensitive credentials. This classic ESPTouch lesson does not provide an authenticated production setup channel. Keep external circuits and other power disconnected.

UNDERSTAND SMARTCONFIG

Follow the two paths

02



Credentials arrive first. Joining the AP comes next.

SmartConfig lets a receiver learn network settings from a nearby sender's Wi-Fi packet pattern. The phone is already on the target router. The board is not joining a temporary setup network in this route.

2.4 GHz

Phone joins this band

Board receives here

5 GHz only

Does not fit this S3 lab

Same name ≠ same band

Use a visible, isolated WPA2-Personal training network.

Do: join the training router's 2.4 GHz network on the phone. Check the band in the phone or router settings. Use a visible WPA2-Personal network, SSID 1–32 UTF-8 bytes, and an 8–63 character printable ASCII password.

A merged 2.4/5 GHz network name can hide which band the phone chose. Select a separate training 2.4 GHz network or have the router administrator help. Do not change an unfamiliar shared router.

PREPARE THE COMPUTER

Build the receiver app

03

Save your files

MicroPython project

→

Install this app

ESP-IDF 5.5.2

The new firmware replaces the MicroPython runtime.

Use idf.py flash; it supplies every image and offset.

1. Save your MicroPython files and private configuration to the computer before changing firmware. For the file backup and return path, use ML-C09 and ML-C10.
2. Install ESP-IDF v5.5.2 for Windows, including the ESP32-S3 tools. Open its **ESP-IDF PowerShell** shortcut.
3. Copy this course's examples/receiver folder to `%USERPROFILE%\MoloLab\ml-c13\receiver`. Keep its `main` folder, CMake files, defaults and partitions file together.

```
Set-Location "$env:USERPROFILE\MoloLab\ml-c13\receiver"
idf.py --version
idf.py set-target esp32s3
idf.py build
```

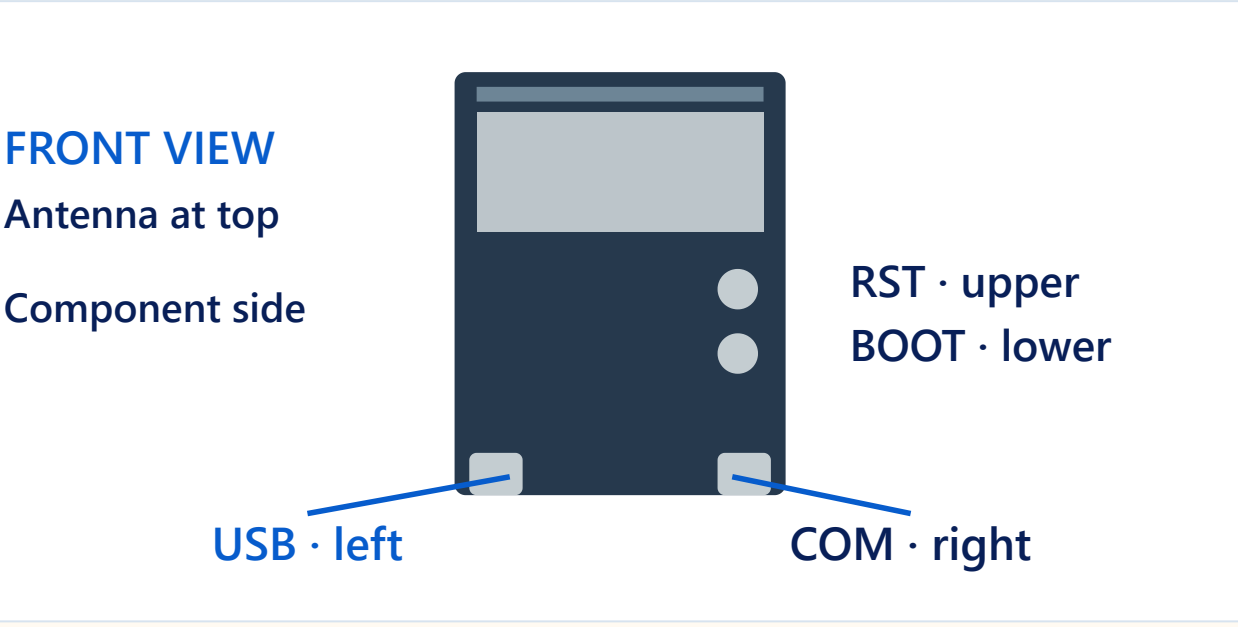
Look for: version 5.5.2, target esp32s3, and a successful build. The supplied defaults select a 16 MB flash, a 2 MB app partition, native USB Serial/JTAG console, and quiet logs. This app does not use PSRAM.

Stop if the version or target is wrong, or the build reports an error. Do not flash an unrelated file to work around a failed build. There is no stock MicroPython `smartconfig()` call in this workflow.

INSTALL THE RECEIVER

Connect, flash, observe

04



1. Connect one USB data cable from the computer to the front-left **USB** socket.
2. If the loader is needed: hold BOOT, press and release RST, then release BOOT. Identify the current download port.
3. Close other serial tools. Replace `COM7` below with that current port.

```
idf.py -p COM7 flash monitor
```

Look for: flash completion, then `LAB READY_CLASSIC_ESPTOUCH_C_TO_CLEAR` and, when no saved configuration works, `LAB LISTEN_CLASSIC_90S`.

Use **Ctrl-]** to exit the monitor. If the runtime port changes, rediscover it, then run `idf.py -p NEW_PORT monitor`. Press RST with BOOT released to start normally.

`idf.py flash` writes the project's bootloader, partition table and app at the generated offsets. Do not write an application-only image at the MicroPython merged-image offset. Keep these instructions with this receiver project.

SEND A TRAINING CANDIDATE

Match the phone sender

05

EspTouch	EspTouch V2
Classic sender	Different protocol
SC_TYPE_ESPTOUCH	Not this receiver

App version 2.4.0 can still offer classic EspTouch.

1. Install Espressif's Android EspTouch app release 2.4.0 from its official project, subject to your phone's installation policy. If installation is unavailable, use page 9.
2. Keep the phone on the target 2.4 GHz training network. Allow the OS permissions the app needs to read Wi-Fi details. Stop if it cannot identify the intended network.
3. Select **EspTouch**, the classic menu item. Do not choose **EspTouch V2**. Enter the training password privately and check the selected network.
4. With the console visible, press RST normally. Start the sender after `LISTEN_CLASSIC_90S`. Follow the classic app's send action. Keep the phone nearby and watch both results.

Observe, do not assume: a phone send animation is not proof that the board received anything. The console provides separate receipt, station-link and acknowledgement checkpoints.

The drawing compares protocol choices; it is not a phone screenshot. An app's release number and the protocol's "V2" name are different things.

VERIFY ONLY WHAT HAPPENED

Read three checkpoints

06

1 · RECEIVED	Configuration decoded Does not prove Wi-Fi
2 · LINK_UP	STA got a nonzero IP SAVED_NVS follows
3 · ACK_SENT	Receiver sent an ACK Check the phone too

STA means the board acts as a station joining a router. **IP address** means a network address assigned to that station. In this app, `LINK_UP` requires the station "got IP" event with a nonzero address.

```
LAB RECEIVED_NOT_CONNECTED
LAB LINK_UP
LAB SAVED_NVS
LAB ACK_SENT_CHECK_PHONE
LAB COMPLETE_VERIFY_RESTART
LAB LISTENER_STOPPED
```

This is a **sample successful trace**; ACK timing can vary. A received candidate may still have a wrong password. A station IP does not prove Internet or cloud access. An ACK sent by the receiver does not prove that the phone displayed it.

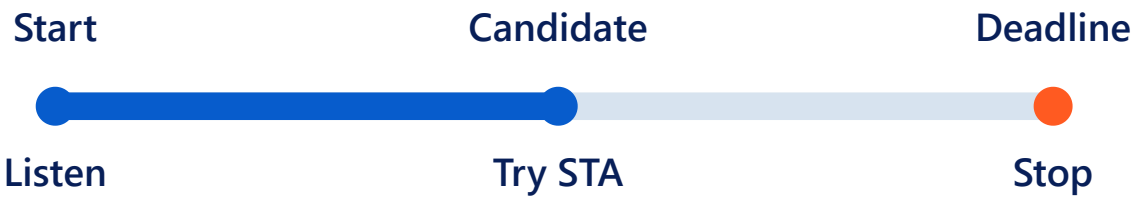
Check the phone's result separately. If the board has `LINK_UP` and `SAVED_NVS` but the phone reports no result, keep that partial outcome: check the phone path and verify restart. Do not label it "nothing received."

TIMEOUT AND RETRY

Give every attempt an end

07

One attempt: at most 90 seconds



No automatic endless setup loop.

The supplied app accepts the first valid classic candidate in a window. It makes one station connection attempt and stops the listener when IP plus saving plus ACK are observed, or when the application's 90-second deadline ends.

1 · Stop sender	Read the last LAB checkpoint
2 · Check one cause	Band / classic mode / private input
3 · Restart once	RST with BOOT released
4 · USB fallback	Read status; c clears only lab key

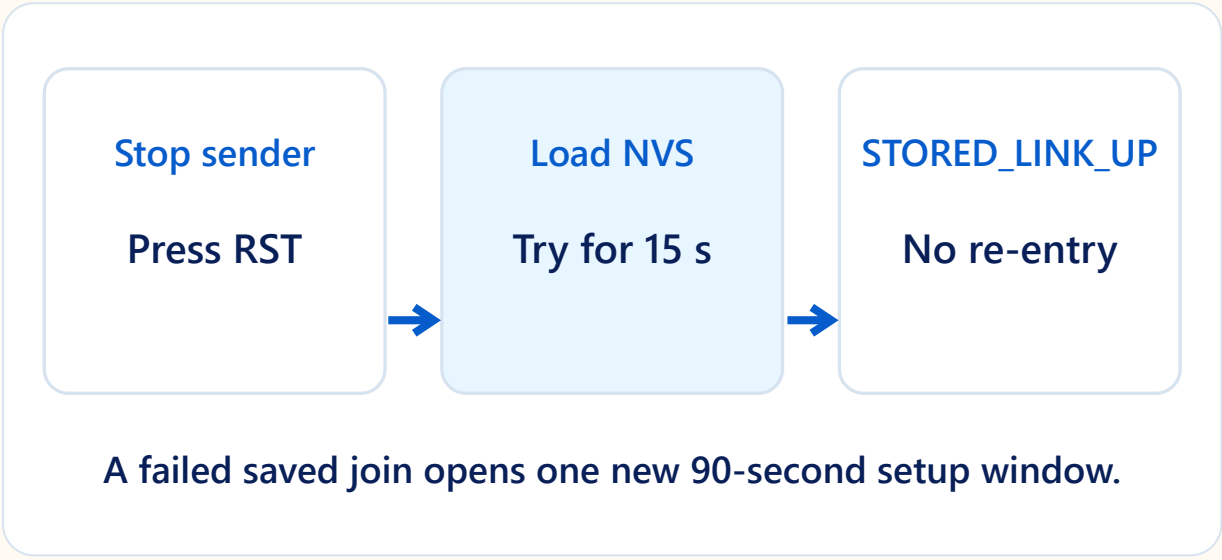
`SCAN_DONE` and `CHANNEL_FOUND` only describe receiver progress. `TIMEOUT_NO_LINK_RESTART_TO_RETRY` means no current station link at the end. Fix one likely cause before a new attempt. If a second controlled attempt fails, use USB to collect the last redacted checkpoint and get help.

A router's isolation or multicast/broadcast handling, a phone permission, the wrong protocol, or the wrong band can block this route. Do not disable network security at random. Try USB configuration instead when the radio path is unsuitable.

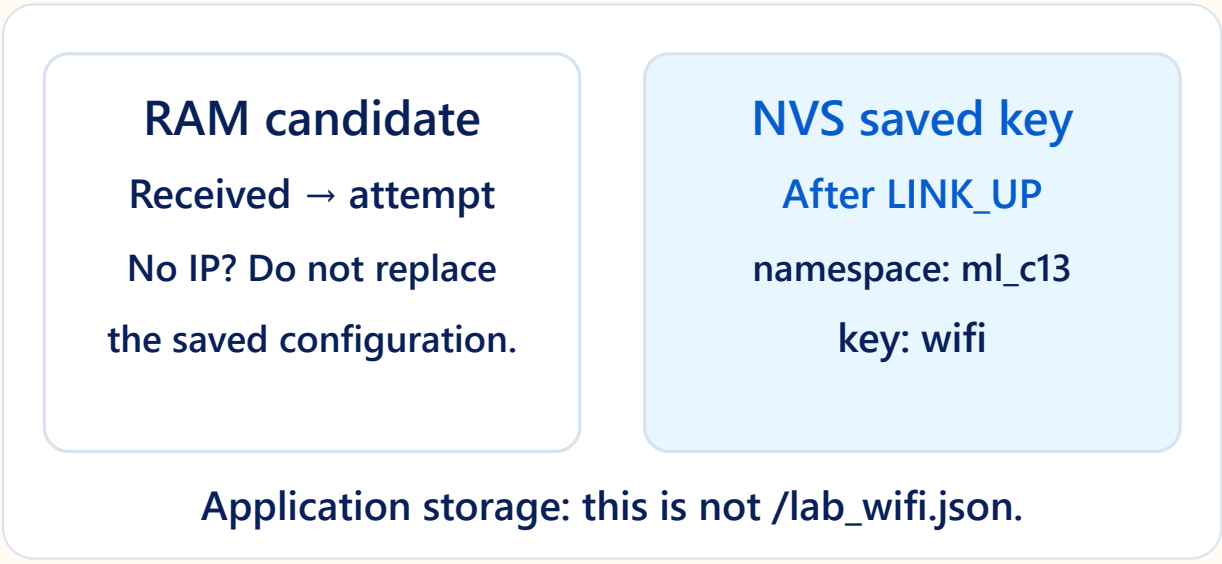
RESTART AND RECOVER

Verify saving; clear narrowly

08



Stop the phone sender. Press RST with BOOT released. A saved configuration is tried for at most 15 seconds; `STORED_LINK_UP` shows this app joined without receiving the settings again. A failed saved join opens one new 90-second setup window.



To forget this lab configuration, type `c` in the IDF monitor; press Enter if your terminal buffers input. `LAB_KEY_CLEARED_RESTART` removes only `ml_c13/wifi` and restarts. The next boot should listen, rather than report `STORED_JOIN_15S`.

`STORAGE_ERROR_GET_HELP` means stop and get help; the clear input is not running in that state. The NVS data is not encrypted by this project. Clearing a key is not secure erasure. Other app data and private backups stay separate. To return to MicroPython, exit the monitor and follow ML-C10, then restore your saved project; this IDF app has no Python REPL.

OFFLINE STATUS EXERCISE

Practice without a radio

09

SCAN_DONE	Still waiting
RECEIVED_NOT_CONNECTED	Decoded, not joined
LINK_UP + SAVED_NVS	Saved link; check ACK
ACK_SENT + listener stopped	Restart to verify saving

Open a computer terminal in this course's `examples` folder. These commands use only included sample states; they do not open a port, send a password or test a board.

```
python trace_lab.py scan-only
python trace_lab.py receipt-only
python trace_lab.py ack-missing
python trace_lab.py complete
```

1. Which case received settings but never proved a station IP?
2. Which case saved a network but still needs a phone-result check?
3. Does the "complete" case prove Internet access or that a restart has succeeded?

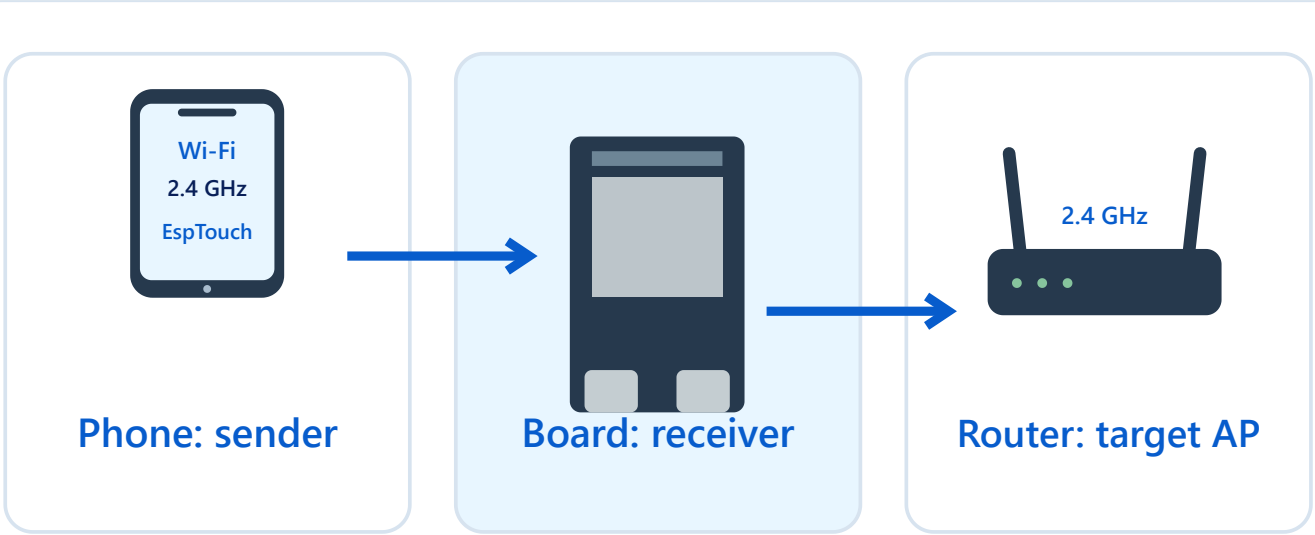
Check your answers: receipt-only; ack-missing; neither. Every successful first setup still needs a normal restart with the sender stopped.

Success: explain what the last checkpoint proves and choose one useful next action. Keep the output labeled `offline_simulation: true`; it is practice, not a device result.

YOUR SUCCESS CHECK

Run a clean final attempt

10



- □ I saved my previous project before installing this firmware.
- □ I can identify USB at front left, RST above BOOT, and the current serial port.
- □ Phone and receiver both use classic EspTouch, with a 2.4 GHz training router.
- □ I distinguish RECEIVED from LINK_UP, SAVED_NVS and ACK_SENT.
- □ I stop at a finite result and check the phone separately.
- □ With the sender stopped, a normal restart shows STORED_LINK_UP.
- □ I can use USB and clear only the application's saved key.

Choose the next route: SmartConfig is useful when a compatible sender and router cooperate. For predictable development or recovery, keep USB available. SoftAP and BLE are other routes with their own receiver firmware and checks.

Take away: keep a short redacted record of the protocol, band, last checkpoint, timeout and next action. Do not include the network password, raw credentials, private screenshots or unrestricted logs.