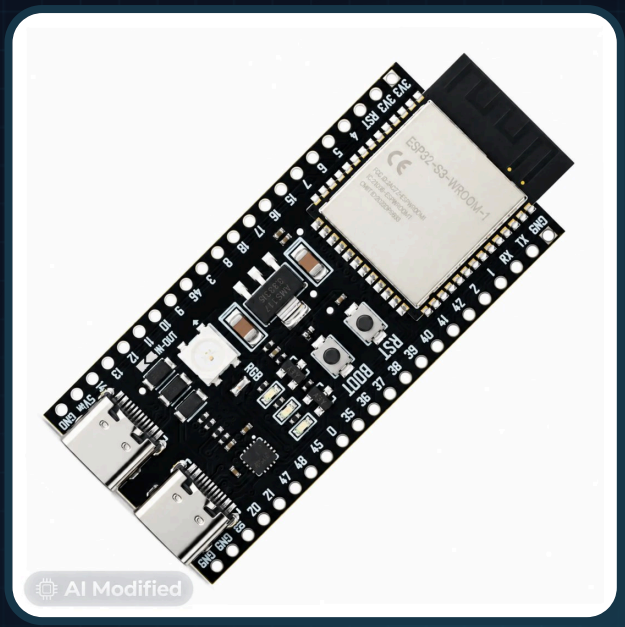
MIKROCONTROLLER · WLAN · BLE · KI

ESP32-S3 N16R8

Der große Bruder des ESP32: Vektorbefehle für KI, 8 MB PSRAM und nativer USB. Erster Upload in rund 20 Minuten, danach sechs Beispiele – bis zu Sprachbefehlen, die ganz ohne Internet erkannt werden.



PROZESSOR	2 × 240 MHz LX7
FLASH	16 MB
PSRAM	8 MB Octal
KI	Vektorbefehle

01 **Einrichten**
IDE, Board-Paket, Einstellungen

02 **Hochladen**
Chip prüfen

03 **Ausprobieren**
PSRAM, Touch, USB, WLAN

04 **KI**
Netze & Sprachbefehle

01 Auf einen Blick

Das Board trägt ein ESP32-S3-WROOM-1 in der größten Speicherausstattung **N16R8**: 16 MB Flash und 8 MB PSRAM. Programmiert wird es wie jeder ESP32 mit der Arduino IDE – nur ein paar Einstellungen sind anders.

Technische Daten

Modul	**ESP32-S3-WROOM-1-N16R8**
Prozessor	**2 × Xtensa LX7, bis 240 MHz, mit Vektorbefehlen**
Speicher	**16 MB Flash (Quad) · 8 MB PSRAM (Octal) · 512 kB SRAM**
Funk	**WLAN 2,4 GHz (b/g/n) · Bluetooth LE 5.0**
USB	**2 × USB-C: COM (USB-UART) und USB (nativ, OTG)**
Versorgung	**5 V über USB-C oder 5Vin, Regler AMS1117-3.3**
Logikpegel	**3,3 V – Pins nicht 5-V-tolerant**
Anschlüsse	**2 × 22 Pins, 2,54-mm-Raster (YD-ESP32-S3-Layout)**

Lieferumfang

1 ×	**ESP32-S3 N16R8 Dev Kit**
1 ×	**Stiftleisten-Set (zum Einlöten)**

! 3,3 V an den Pins
Die GPIOs vertragen keine 5 V. 5-V-Signale nur über Spannungsteiler oder Pegelwandler anschließen.

i Stiftleisten einlöten
Die beiden 22-poligen Leisten liegen bei. Mit der langen Seite nach unten ins Steckbrett stecken, Board auflegen, alle Pins verlöten.

ESP32 oder ESP32-S3?

	ESP32 DEVKIT	ESP32-S3 N16R8 (DIESES BOARD)
Prozessor	2 × LX6, 240 MHz	2 × LX7, 240 MHz, **Vektorbefehle für KI**
Speicher	4 MB Flash, 520 kB SRAM	**16 MB Flash, 8 MB PSRAM**, 512 kB SRAM
Bluetooth	Classic + LE	nur LE 5.0
USB	nur über Wandler	Wandler **und nativer USB** (Tastatur, MIDI …)

Was du zusätzlich brauchst

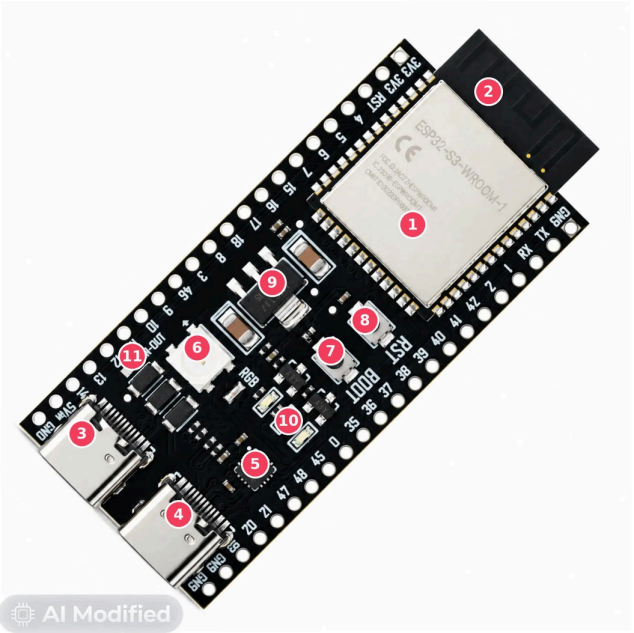
USB-C-Datenkabel
Kein reines Ladekabel – es muss Daten übertragen

PC mit Arduino IDE 2
Windows, macOS oder Linux, kostenlos

Für die KI-Sprachsteuerung
INMP441 I2S-Mikrofon + Steckbrett, Jumper
MKR-INMP441-64

02 Dein Board im Detail

Das Board folgt dem verbreiteten YD-ESP32-S3-Layout: zwei USB-C-Buchsen, RGB-LED, zwei Tasten und 2 × 22 Pins. Die Nummern im Foto gehören zur Tabelle.



1. **ESP32-S3-WROOM-1** – Das Funkmodul mit Prozessor, 16 MB Flash und 8 MB PSRAM unter dem Blechdeckel. Die Variante steht auf dem Etikett bzw. Karton: **N16R8**.
2. **PCB-Antenne** – WLAN und Bluetooth. Kein Metall darüber oder darunter, nicht auf leitende Flächen legen.
3. **USB-C-Buchse** – Eine der beiden Buchsen heißt **COM** – über den USB-UART-Wandler (5) zum seriellen Monitor. Damit lädst du hoch.
4. **USB-C-Buchse** – Die andere heißt **USB** – direkt am nativen USB des S3 (GPIO19/20). Für USB-Tastatur, MIDI, Massenspeicher.
5. **USB-UART-Wandler** – Macht aus „COM“ einen seriellen Port und schaltet beim Hochladen automatisch in den Boot-Modus.
6. **RGB-LED** – Adressierbare WS2812 an **GPIO48**. Ansteuerung mit `rgbLedWrite(48, r, g, b)`.
7. **Taste BOOT** – Hängt an GPIO0. Gedrückt beim Einschalten: Download-Modus. Im Betrieb als Taster nutzbar.
8. **Taste RST** – Neustart des Sketches.
9. **Spannungsregler** – AMS1117 macht aus 5 V die 3,3 V für Chip und den 3V3-Pin.
10. **Status-LEDs** – Power sowie die Datenleitungen TX/RX des COM-Ports.
11. **Lötbrücke IN-OUT** – Offen ist 5Vin nur ein Eingang. Geschlossen liegt die USB-Spannung auch an 5Vin an – nur schließen, wenn du 5 V abgreifen willst.

i Welche Buchse wofür?

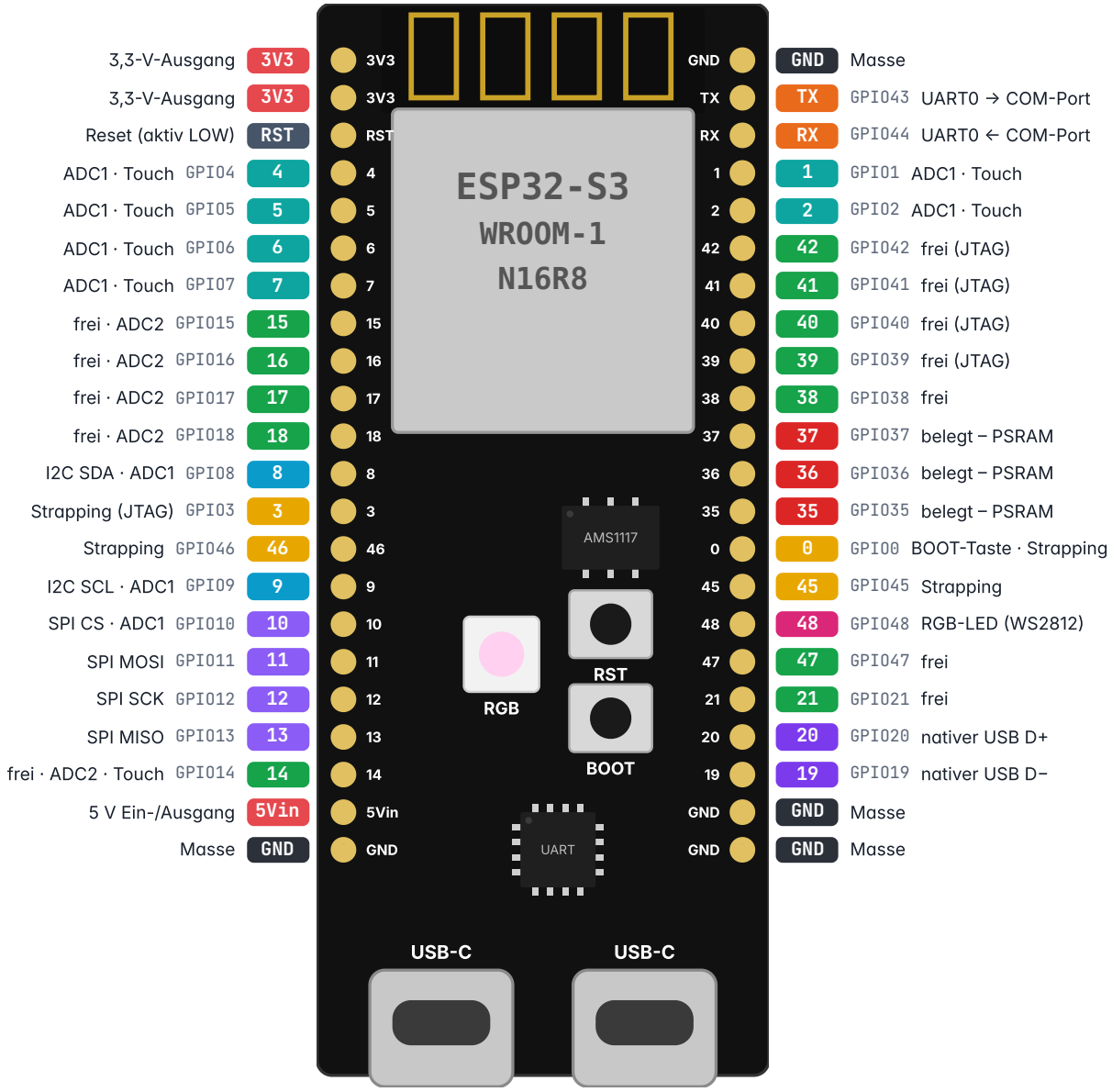
Hochladen und serieller Monitor: immer **COM**. Die Buchse **USB** brauchst du erst, wenn der S3 selbst ein USB-Gerät sein soll (Beispiel 3). Beide Buchsen versorgen das Board – ein Kabel genügt.

! Drei Pins sind tabu

GPIO35, GPIO36 und GPIO37 stehen auf der Leiste, sind beim N16R8 aber intern mit dem Octal-PSRAM verbunden. Dort nichts anschließen.

03 Pinbelegung

Antenne oben, USB-Buchsen unten. Auf der Platine steht nur die Nummer – im Code schreibst du genau diese Zahl, z. B. `digitalWrite(21, HIGH)`.



Versorgung · Masse · frei nutzbar · ADC1 · Touch · I2C · SPI · UART0 / COM · nativer USB · Strapping – Vorsicht · nicht nutzbar · RGB-LED

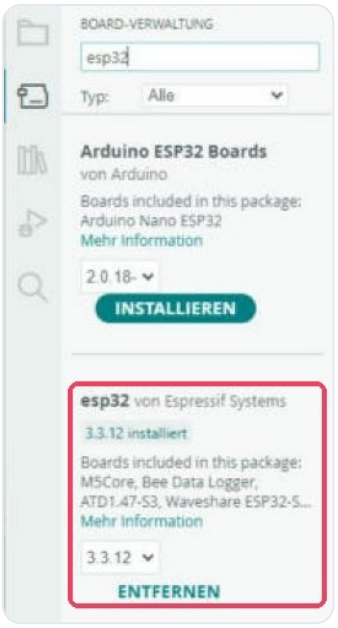
04 Einrichten

Die Schritte 1 und 2 sind dieselben wie beim ESP32. Entscheidend beim S3 sind die Einstellungen im Menü `Werkzeuge` – vor allem **PSRAM**.

1 Arduino IDE installieren

Aktuelle Arduino IDE 2 von `arduino.cc/en/software` laden und installieren.

2 ESP32-Board-Paket installieren



Links auf das Board-Symbol klicken, **esp32** eingeben und beim Eintrag **esp32 von Espressif Systems** auf **Installieren** klicken (rund 300 MB). Version 3.0 oder neuer ist Pflicht – die KI-Beispiele brauchen sie. Nicht „Arduino ESP32 Boards“ nehmen.

Das Bild stammt aus unserem ESP32-Guide; der Eintrag ist für ESP32 und ESP32-S3 derselbe.

3 Board anschließen, Board und Port wählen

USB-C-Kabel an die Buchse **COM**. In der Board-Auswahl **ESP32S3 Dev Module** suchen und den neuen COM-Port wählen. Kein Port? Siehe Seite 15.

4 Einstellungen unter Werkzeuge

Einmal setzen, die IDE merkt sie sich. Nicht genannte Punkte bleiben auf dem Standardwert.

MENÜPUNKT	WERT	WARUM
USB CDC On Boot	`Disabled`	Serielle Ausgabe läuft über die COM-Buchse.
Flash Mode	`QIO 80MHz`	Der Flash des N16R8 ist Quad-SPI.
Flash Size	`16MB (128Mb)`	Sonst nutzt der Sketch nur 4 MB.
Partition Scheme	`16M Flash (3MB APP/9.9MB FATFS)`	3 MB für den Sketch. Für Beispiel 6: „ESP SR 16M …“.
PSRAM	`OPI PSRAM`	Wichtig! Das „R8“ ist Octal-PSRAM. Falsch gewählt: 0 MB PSRAM oder Neustart-Schleife.
USB Mode	`Hardware CDC and JTAG`	Standard. Für Beispiel 3: „USB-OTG (TinyUSB)“.

05 Hochladen & Chip prüfen

Der erste Sketch liest aus, was wirklich auf deinem Board steckt – Chip, Flash und PSRAM – und lässt die RGB-LED durch die Farben laufen.

MAKEROO_ESP32S3_Start.ino

```ino
// MAKEROO · ESP32-S3 N16R8 · Erster Start: Chip pruefen und RGB-LED
#include "esp_flash.h"

const int RGB_PIN = 48;                          // WS2812 auf dem Board ("RGB")

void setup() {
  Serial.begin(115200);
  delay(1000);
  uint32_t flash = 0;
  esp_flash_get_physical_size(NULL, &flash);     // echte Chipgroesse
  Serial.println("\nMAKEROO ESP32-S3 Dev Kit");
  Serial.printf("Chip:   %s, Revision %d, %d Kerne, %d MHz\n", ESP.getChipModel(),
                ESP.getChipRevision(), ESP.getChipCores(), (int)ESP.getCpuFreqMHz());
  Serial.printf("Flash:  %d MB\n", (int)(flash / (1024 * 1024)));
  Serial.printf("PSRAM:  %d MB (frei: %d kB)\n", (int)(ESP.getPsramSize() / (1024 * 1024)),
                (int)(ESP.getFreePsram() / 1024));
  Serial.printf("RAM:    %d kB frei im internen Speicher\n", (int)(ESP.getFreeHeap() / 1024));
  if (ESP.getPsramSize() == 0) {
    Serial.println("Kein PSRAM! Werkzeuge -> PSRAM -> \"OPI PSRAM\" waehlen.");
  }
}

void loop() {
  const uint8_t farbe[5][3] = {{40, 0, 0}, {0, 40, 0}, {0, 0, 40}, {25, 25, 25}, {0, 0, 0}};
  const char* name[5] = {"rot", "gruen", "blau", "weiss", "aus"};
  for (int i = 0; i < 5; i++) {
    rgbLedWrite(RGB_PIN, farbe[i][0], farbe[i][1], farbe[i][2]);
    Serial.printf("RGB-LED: %s\n", name[i]);
    delay(800);
  }
}
```

Kompiliert für „ESP32S3 Dev Module“: 298 kB Flash (9 %), 22 kB RAM (6 %).

Zeile 4
RGB-LED
Die WS2812-LED „RGB“ hängt an GPIO48.

Zeile 10
Echte Flash-Größe
Liest die Größe aus dem Chip – unabhängig von der Einstellung in der IDE.

Zeile 15–16
PSRAM prüfen
8 MB erwartet. 0 MB heißt: PSRAM-Einstellung falsch.

SO SIEHT DIE AUSGABE AUS · 115200 BAUD

```
MAKEROO ESP32-S3 Dev Kit
Chip:   ESP32-S3, Revision x, 2 Kerne, 240 MHz
Flash:  16 MB
PSRAM:  8 MB (frei: ~8 100 kB)
RAM:    ~3xx kB frei im internen Speicher
RGB-LED: rot
RGB-LED: gruen
```

Hochladen

Pfeil-Symbol oder `Strg` + `U`. Das erste Kompilieren dauert etwa eine Minute. Danach den Monitor öffnen (`Strg` + `Umschalt` + `M`) und 115200 Baud einstellen; ggf. **RST** drücken.

Was die Zahlen bedeuten

16 MB und **8 MB** bestätigen ein N16R8. Revision und freier RAM schwanken je nach Charge und Board-Paket.

06 Beispiele: Was der S3 kann

Beispiel 1

8 MB PSRAM nutzen

KEIN ZUBEHÖR

Der interne Arbeitsspeicher hat 512 kB. Mit `ps_malloc()` holst du dir Speicher aus dem 8-MB-PSRAM – hier einen 4-MB-Puffer, wie ihn Kamerabilder, Audio oder KI-Modelle brauchen.

S3_01_PSRAM.ino

```ino
// MAKEROO · ESP32-S3 · Beispiel 1: 8 MB PSRAM nutzen
void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.printf("PSRAM gesamt: %d kB, frei: %d kB\n",
                (int)(ESP.getPsramSize() / 1024), (int)(ESP.getFreePsram() / 1024));
  const size_t GROESSE = 4 * 1024 * 1024;       // 4 MB - zu gross fuer den internen RAM
  uint8_t* puffer = (uint8_t*)ps_malloc(GROESSE);
  if (puffer == nullptr) {
    Serial.println("ps_malloc fehlgeschlagen - PSRAM aktiviert?");
    return;
  }
  uint32_t t0 = millis();
  for (size_t i = 0; i < GROESSE; i++) puffer[i] = i & 0xFF;   // schreiben
  uint32_t t1 = millis();
  uint32_t summe = 0;
  for (size_t i = 0; i < GROESSE; i++) summe += puffer[i];    // lesen
  uint32_t t2 = millis();
  Serial.printf("4 MB geschrieben in %d ms, gelesen in %d ms\n", (int)(t1 - t0), (int)(t2 - t1));
  Serial.printf("Pruefsumme: %u (erwartet 534773760)\n", (unsigned)summe);
  Serial.printf("PSRAM frei mit Puffer: %d kB\n", (int)(ESP.getFreePsram() / 1024));
  free(puffer);
}

void loop() {}
```

Kompiliert für „ESP32S3 Dev Module“: 273 kB Flash (8 %), 22 kB RAM (6 %).

BEISPIELAUSGABE · 115200 BAUD

```
PSRAM gesamt: ~8 190 kB, frei: ~8 100 kB
4 MB geschrieben in … ms, gelesen in … ms
Pruefsumme: 534773760 (erwartet 534773760)
PSRAM frei mit Puffer: ~4 000 kB
```

Gut zu wissen

Stimmt die Prüfsumme, wurde jedes Byte richtig gespeichert. PSRAM ist langsamer als der interne RAM – zeitkritische Daten bleiben besser dort.

Wofür braucht man so viel Speicher?

Kamerabild

Ein Bild mit 320 × 240 Pixeln in 16-Bit-Farbe belegt 150 kB – mehr als ein Viertel des internen RAM. Im PSRAM passen über 50 davon.

Audio

10 Sekunden Sprache mit 16 kHz und 16 Bit sind 313 kB. Spracherkennung (Beispiel 6) puffert ihr Audio genau dort.

Display

Ein Bildspeicher für ein 480 × 320-Display braucht 300 kB – damit zeichnest du flimmerfrei im Hintergrund.

So landen Daten im PSRAM

`ps_malloc(größe)` oder `heap_caps_malloc(größe, MALLOC_CAP_SPIRAM)` statt `malloc()`. Große Puffer nicht als globale Arrays anlegen – die landen im internen RAM. Viele Bibliotheken (Kamera, TensorFlow Lite, LVGL) nutzen PSRAM automatisch, wenn er aktiviert ist.

Beispiel 2 Berührung erkennen

DRAHT ODER ALUFOLIE

GPIO1 bis GPIO14 sind Touch-Eingänge. Ein Draht an GPIO4 genügt – bei Berührung leuchtet die RGB-LED grün.

S3_02_Touch.ino

```ino
// MAKEROO · ESP32-S3 · Beispiel 2: Beruehrung erkennen
const int TOUCH_PIN = 4;          // Touch T4: Draht oder Alufolie
const int RGB_PIN = 48;
uint32_t ruhe;

void setup() {
  Serial.begin(115200);
  delay(1000);
  ruhe = touchRead(TOUCH_PIN);      // beim Start nicht beruehren!
  Serial.printf("Ruhewert: %u\n", (unsigned)ruhe);
}

void loop() {
  uint32_t wert = touchRead(TOUCH_PIN);
  bool beruehrt = wert > ruhe * 1.2; // beim S3 STEIGT der Wert
  rgbLedWrite(RGB_PIN, 0, beruehrt ? 40 : 0, 0);
  Serial.printf("%u%s\n", (unsigned)wert, beruehrt ? "  <- beruehrt" : "");
  delay(200);
}
```

Kompiliert für „ESP32S3 Dev Module“: 314 kB Flash (10 %), 22 kB RAM (7 %).

Ergebnis

Im Monitor läuft der Messwert mit, zuerst der Ruhewert. Berührst du den Draht, steigt die Zahl deutlich, dahinter steht „← beruehrt“ und die RGB-LED leuchtet grün.

Anders als beim ESP32

Beim S3 **steigt** der Messwert bei Berührung, beim alten ESP32 sinkt er. Der Sketch merkt sich beim Start den Ruhewert – dabei den Draht nicht berühren.

Beispiel 3 Das Board als USB-Tastatur

ZWEITES USB-C-KABEL

Der S3 hat USB direkt im Chip. Damit meldet er sich am PC als Tastatur – ein Druck auf **BOOT** tippt einen Text. So baust du Makro-Tasten, Fußschalter oder Stream-Decks.

S3_03_USB_Tastatur.ino

```ino
// MAKEROO · ESP32-S3 · Beispiel 3: Das Board als USB-Tastatur
// Werkzeuge -> USB Mode: "USB-OTG (TinyUSB)". Danach Kabel an die Buchse "USB".
#include "USB.h"
#include "USBHIDKeyboard.h"

USBHIDKeyboard tastatur;
const int TASTE = 0;                          // die BOOT-Taste

void setup() {
  pinMode(TASTE, INPUT_PULLUP);
  tastatur.begin();
  USB.begin();                                // meldet sich am PC als Tastatur
}

void loop() {
  if (digitalRead(TASTE) == LOW) {
    tastatur.print("Hallo von MAKEROO! ");
    delay(300);
    while (digitalRead(TASTE) == LOW) delay(10); // warten bis losgelassen
  }
}
```

Kompiliert für „ESP32S3 Dev Module“ · USB Mode: USB-OTG (TinyUSB): 357 kB Flash (11 %), 54 kB RAM (16 %).

So testest du

1. `Werkzeuge → USB Mode → USB-OTG (TinyUSB)`.
2. Über **COM** hochladen.
3. Zweites Kabel an **USB** – der PC meldet eine neue Tastatur.
4. Editor öffnen, **BOOT** drücken.

Achtung

Die Tastatur tippt in jedes Fenster, das gerade aktiv ist. Erst einen leeren Editor öffnen.

Mehr

Im Board-Paket stecken weitere USB-Beispiele: Maus, Gamepad, MIDI und USB-Stick (`Datei → Beispiele → USB`).

Beispiel 4

Farbe per Webseite

SMARTPHONE

Der S3 öffnet sein eigenes WLAN „MAKEROO-S3“ und liefert eine Seite mit Farbwähler. Die RGB-LED übernimmt sofort jede gewählte Farbe.

S3_04_Webserver.ino

```ino
// MAKEROO · ESP32-S3 · Beispiel 4: RGB-LED per Webseite
#include <WiFi.h>
#include <WebServer.h>

const int RGB_PIN = 48;
WebServer server(80);

const char SEITE[] = R"(<!doctype html><meta name=viewport content="width=device-width">
<body style="font-family:sans-serif;text-align:center"><h2>MAKEROO ESP32-S3</h2>
<p>Farbe waehlen:</p><input type=color id=c value="#00c0ff" style="width:80%;height:50vh">
<script>c.oninput=()=>fetch('/farbe?c='+c.value.slice(1))</script>)";

void setup() {
  Serial.begin(115200);
  WiFi.softAP("MAKEROO-S3", "makeroo123");      // eigenes WLAN
  Serial.print("Hotspot aktiv: http://");
  Serial.println(WiFi.softAPIP());
  server.on("/", [] { server.send(200, "text/html", SEITE); });
  server.on("/farbe", [] {
    long c = strtol(server.arg("c").c_str(), nullptr, 16);   // "ff8800" -> Zahl
    rgbLedWrite(RGB_PIN, (c >> 16) / 4, ((c >> 8) & 0xFF) / 4, (c & 0xFF) / 4);
    server.send(200, "text/plain", "ok");
  });
  server.begin();
}

void loop() {
  server.handleClient();
}
```

Kompiliert für „ESP32S3 Dev Module“: 889 kB Flash (28 %), 46 kB RAM (14 %).



Seite aus dem Sketch

So testest du

1. Hochladen.
2. Am Handy WLAN **MAKEROO-S3**, Passwort **makeroo123**.
3. Im Browser `192.168.4.1` öffnen.
4. Auf die Farbfläche tippen und eine Farbe wählen.

BEISPIELAUSGABE · 115200 BAUD

```
Hotspot aktiv: http://192.168.4.1
```

Gut zu wissen

Die Farbwerte werden durch 4 geteilt – volle Helligkeit blendet. Passwort vor dem Dauerbetrieb ändern.

07 KI auf dem ESP32-S3 – was geht wirklich?

KI auf einem Mikrocontroller heißt: Ein kleines, fertig trainiertes neuronales Netz läuft direkt auf dem Chip – ohne Cloud, ohne Internet, in Millisekunden. Trainiert wird am PC, der S3 führt das Modell aus. Fachleute nennen das **TinyML** oder Edge-KI.

Vektorbefehle

Die LX7-Kerne verarbeiten mehrere 8-Bit-Werte in einem Befehl – genau die Rechnung, aus der neuronale Netze bestehen. Espressifs Bibliothek ESP-NN nutzt das. Laut Espressif sinkt die Rechenzeit eines Personen-Erkennungsmodells damit von 2,3 s auf 54 ms (240 MHz).

Speicher

8 MB PSRAM für Modelle, Audio- und Bildpuffer, 16 MB Flash für die Modelldateien. Allein das Sprachmodell aus Beispiel 6 belegt rund 3,3 MB.

Zwei Kerne

Ein Kern rechnet die KI, der andere kümmert sich um WLAN, Display und Sensoren.

Ehrlich gesagt

ChatGPT & Co. laufen nicht auf dem Chip – solche Modelle sind tausendfach größer. Der S3 kann sie aber per WLAN als Cloud-Dienst nutzen.

Was geht mit diesem Board?

AUFGABE	MACHBAR?	WERKZEUG
Aktivierungswort + Sprachbefehle, offline	**Ja** – mit INMP441	ESP-SR im Board-Paket (Beispiel 6), ESPHome *micro_wake_word*
Kleine eigene Netze	**Ja**	TensorFlow Lite Micro (Beispiel 5)
Sensordaten deuten: Gesten, Vibration, Anomalien	**Ja** – mit Sensor, z. B. MPU-6050	Edge Impulse, TensorFlow Lite Micro
Geräusche erkennen (Klatschen, Glas, Hund)	**Ja** – mit INMP441	Edge Impulse
Bilder: Personen, Gesichter, Objekte	**Nur mit Kamera** – dieses Board hat keinen Kameraanschluss	ESP-WHO, ESP-DL auf einem S3-Kameraboard
Große Sprachmodelle (ChatGPT, Claude …)	**Nicht lokal** – nur per WLAN-API	HTTPS-Anfrage an einen Cloud-Dienst
Modelle trainieren	**Nein** – am PC oder in der Cloud	Python/TensorFlow, Edge Impulse

So kommt ein Modell auf den Chip

1 **Daten sammeln**
Sensorwerte oder Audio aufnehmen – oft mit dem S3 selbst.

2 **Trainieren**
Am PC oder im Browser (Edge Impulse).

3 **Verkleinern**
Quantisieren: 32-Bit-Kommazahlen werden 8-Bit-Ganzzahlen.

4 **Exportieren**
Als C-Array (`model.h`) oder Arduino-Bibliothek.

5 **Ausführen**
Der S3 rechnet das Netz – die „Inferenz“.

KI-Beispiel 5

Ein neuronales Netz rechnet sin(x)

KEIN ZUBEHÖR

Das „Hallo Welt“ von TensorFlow Lite Micro: Ein winziges Netz (3 Schichten, 2,4 kB) hat gelernt, die Sinuskurve nachzubilden. Der S3 rechnet es für 13 Werte und vergleicht mit dem echten Sinus. Die RGB-LED pulsiert im Takt der Kurve.

S3_05_KI_Sinus.ino

```ino
// MAKEROO · ESP32-S3 · KI-Beispiel 5: Ein neuronales Netz rechnet sin(x)
// Bibliothek "Chirale_TensorFlowLite" installieren, model.h liegt im Sketch-Ordner
#include <Chirale_TensorFlowLite.h>
#include "model.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"

const int RGB_PIN = 48;
alignas(16) uint8_t arena[2048];                // Arbeitsspeicher des Netzes
tflite::MicroInterpreter* netz;
TfLiteTensor *ein, *aus;

void setup() {
  Serial.begin(115200);
  static tflite::AllOpsResolver ops;
  static tflite::MicroInterpreter interpreter(tflite::GetModel(g_model), ops, arena, sizeof(arena));
  netz = &interpreter;
  if (netz->AllocateTensors() != kTfLiteOk) {
    Serial.println("Arena zu klein!");
    while (true) delay(10);
  }
  ein = netz->input(0);
  aus = netz->output(0);
}

void loop() {
  for (float x = 0; x < 6.3; x += 0.5) {
    ein->data.int8[0] = x / ein->params.scale + ein->params.zero_point;   // quantisieren
    uint32_t t = micros();
    netz->Invoke();                                                       // Inferenz
    t = micros() - t;
    float y = (aus->data.int8[0] - aus->params.zero_point) * aus->params.scale;
    Serial.printf("x = %.1f   KI: %+.2f   echt: %+.2f   (%d us)\n", x, y, sin(x), (int)t);
    rgbLedWrite(RGB_PIN, 0, 0, (uint8_t)((y + 1) * 30));                  // Helligkeit folgt
    delay(300);
  }
}
```

Kompiliert für „ESP32S3 Dev Module“ · mit Chirale_TensorFlowLite 2.0.0: 577 kB Flash (18 %), 31 kB RAM (9 %).

Ergebnis des Netzes

X	KI	ECHT
0.0	+0.00	+0.00
0.5	+0.48	+0.48
1.0	+0.88	+0.84
1.5	+1.04	+1.00
2.0	+0.88	+0.91
2.5	+0.61	+0.60
3.0	+0.13	+0.14
3.5	-0.31	-0.35
4.0	-0.79	-0.76
4.5	-1.02	-0.98
5.0	-0.99	-0.96
5.5	-0.66	-0.71
6.0	-0.33	-0.28

Vorbereitung

1. Bibliotheksverwalter: **Chirale_TensorFlowLite** installieren.
2. Sketch-Ordner aus dem Download öffnen – `model.h` muss daneben liegen.
3. Hochladen, Monitor auf 115200 Baud.

Was du siehst

Jede Zeile zeigt x, die KI-Antwort, den echten Sinus und die Rechenzeit in Mikrosekunden. Die KI-Werte in der Tabelle haben wir mit demselben Modell am PC nachgerechnet.

Quantisierung

Das Netz rechnet mit 8-Bit-Ganzzahlen statt Kommazahlen – viermal kleiner und schneller. Dafür liegt es um ein paar Hundertstel daneben. Die beiden Zeilen mit `params.scale` rechnen hin und zurück.

Schneller geht es

Diese Bibliothek nutzt allgemeine Rechenfunktionen. Mit Espressifs *esp-tflite-micro* (ESP-IDF) kommen die Vektorbefehle über ESP-NN dazu – bei größeren Modellen ein Vielfaches schneller.

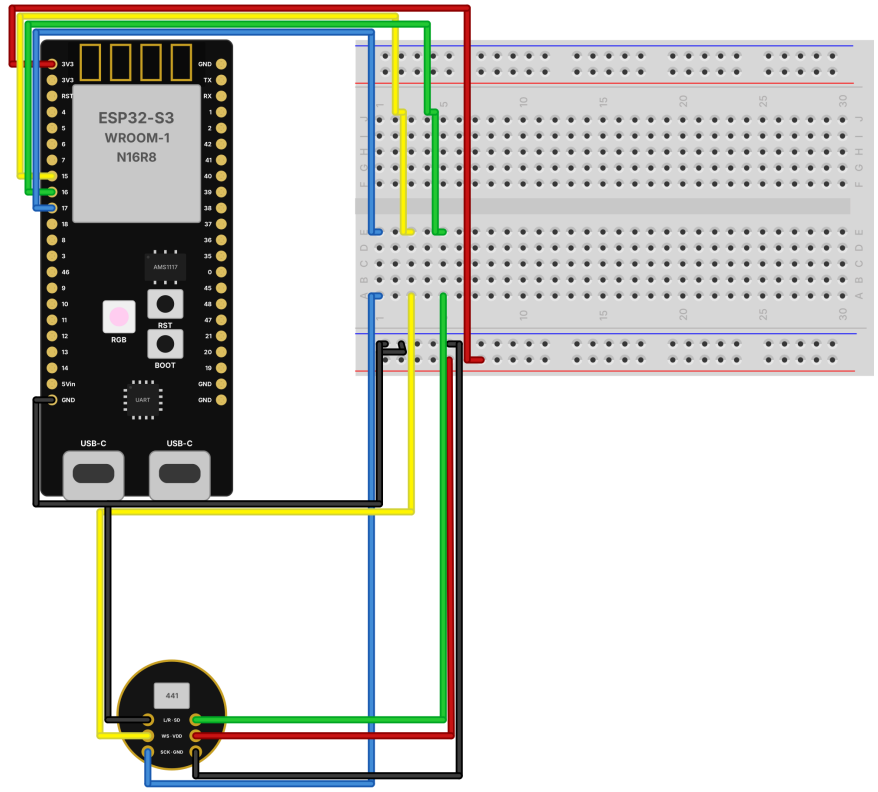
KI-Beispiel 6

Sprachbefehle ohne Internet

INMP441-MIKROFON

Mit dem Spracherkenner ESP-SR, der im Board-Paket steckt, hört der S3 auf das Aktivierungswort **„Hi ESP“** und versteht danach englische Befehle. Zwei neuronale Netze laufen dabei auf dem Chip: WakeNet für das Aktivierungswort, MultiNet für die Befehle. Keine Cloud, kein Konto.

STECKPLATINEN-ANSICHT · FRITZING



INMP441		ESP32-S3
VDD	→	3V3
GND	→	GND
L/R	→	GND
SCK	→	17
WS	→	15
SD	→	16

Zwei Einstellungen sind Pflicht

`Partition Scheme → ESP SR 16M (3MB APP/6MB SPIFFS/3.9MB MODEL)` – sonst fehlt das Sprachmodell. Und `PSRAM → OPI PSRAM`. Die IDE lädt das 3,3-MB-Modell beim Hochladen automatisch mit.

L/R an GND

Damit liefert das Mikrofon den linken Kanal – genau den liest der Sketch. Kabel kurz halten (unter 15 cm).

S3_06_KI_Sprache.ino

```ino
// MAKEROO · ESP32-S3 · KI-Beispiel 6: Sprachbefehle offline (INMP441)
// Werkzeuge -> Partition Scheme: "ESP SR 16M ...", PSRAM: "OPI PSRAM"
#include "ESP_I2S.h"
#include "ESP_SR.h"

const int RGB_PIN = 48;
const int MIC_SCK = 17, MIC_WS = 15, MIC_SD = 16;   // INMP441, L/R an GND
I2SClass i2s;

enum { AN, AUS, ROT, GRUEN, BLAU };
const char* NAME[] = {"Licht an", "Licht aus", "Rot", "Gruen", "Blau"};
static const sr_cmd_t befehle[] = {
  {AN, "Turn on the light"}, {AN, "Lights on"}, {AUS, "Turn off the light"},
  {AUS, "Lights off"}, {ROT, "Set color red"}, {GRUEN, "Set color green"},
  {BLAU, "Set color blue"},
};

void ereignis(sr_event_t e, int befehl, int phrase) {
  if (e == SR_EVENT_WAKEWORD) {                // "Hi ESP" gehoert
    Serial.println("Hi ESP erkannt - sag einen Befehl");
    ESP_SR.setMode(SR_MODE_COMMAND);
  } else if (e == SR_EVENT_TIMEOUT) {          // Befehl kam nicht rechtzeitig
    Serial.println("Zeit abgelaufen - warte auf \"Hi ESP\"");
    ESP_SR.setMode(SR_MODE_WAKEWORD);
  } else if (e == SR_EVENT_COMMAND) {
    Serial.printf("Befehl: %s\n", NAME[befehl]);
    const uint8_t f[5][3] = {{30, 30, 30}, {0, 0, 0}, {40, 0, 0}, {0, 40, 0}, {0, 0, 40}};
    rgbLedWrite(RGB_PIN, f[befehl][0], f[befehl][1], f[befehl][2]);
    ESP_SR.setMode(SR_MODE_COMMAND);           // weitere Befehle erlaubt
  }
}

void setup() {
  Serial.begin(115200);
  i2s.setPins(MIC_SCK, MIC_WS, -1, MIC_SD);
  i2s.setTimeout(1000);
  i2s.begin(I2S_MODE_STD, 16000, I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_MODE_MONO, I2S_STD_SLOT_LEFT);
  i2s.configureRX(16000, I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_MODE_MONO, I2S_RX_TRANSFORM_32_TO_16);
  ESP_SR.onEvent(ereignis);
  ESP_SR.begin(i2s, befehle, sizeof(befehle) / sizeof(sr_cmd_t), SR_CHANNELS_MONO, SR_MODE_WAKEWORD, "M");
  Serial.println("Sag \"Hi ESP\" ...");
}

void loop() {
  delay(100);                                  // alles laeuft im Hintergrund
}
```

Kompiliert für „ESP32S3 Dev Module“ · Partition Scheme: ESP SR 16M: 2393 kB Flash (77 %), 40 kB RAM (12 %).

Befehle im Sketch

`Hi ESP`	aufwecken
`Turn on the light`	Licht an (weiß)
`Turn off the light`	Licht aus
`Set color red`	rot
`Set color green`	grün
`Set color blue`	blau

BEISPIELAUSGABE · 115200 BAUD

```
Sag "Hi ESP" ...
Hi ESP erkannt - sag einen Befehl
Befehl: Licht an
Befehl: Blau
Zeit abgelaufen -
 warte auf "Hi ESP"
```

So klappt es

Deutlich, mit normaler Lautstärke aus 0,5 bis 2 m sprechen. „Hi ESP“ englisch: „Hai I-Es-Pi“. Ruhige Umgebung hilft.

Eigene Befehle

Einfach Zeilen in die Liste `befehle[]` eintragen – auf Englisch, mit eigener Nummer. MultiNet erkennt sie ohne neues Training. Deutsche Befehle unterstützt dieses Modell nicht.

Mehr Sprache

Ein Aktivierungswort wie „Okay Nabu“ und Befehle in Deutsch bekommst du mit ESPHome + Home Assistant (Seite 14).

08 KI: Wie geht's weiter?

Eigene Modelle mit Edge Impulse

Die bequemste Art, eigene KI zu trainieren – im Browser, ohne Programmierkenntnisse in Python.

1 Projekt anlegen

Kostenlos auf `edgeimpulse.com`.

2 Daten aufnehmen

Z. B. Bewegungen mit dem MPU-6050 oder Geräusche mit dem INMP441 – je Klasse ein paar Minuten.

3 Trainieren

Im Browser ein Modell wählen und trainieren lassen. Edge Impulse zeigt Genauigkeit und Speicherbedarf.

4 Als Arduino-Bibliothek exportieren

Die ZIP-Datei über `Sketch → Bibliothek einbinden → .ZIP-Bibliothek hinzufügen` installieren und das mitgelieferte ESP32-Beispiel öffnen.

Sprachassistent mit ESPHome

Mit dem INMP441 wird der S3 zum lokalen Sprachassistenten für Home Assistant: Das Aktivierungswort erkennt *micro_wake_word* auf dem Chip, die Sprache verarbeitet dein Home-Assistant-Server. Grundgerüst für dieses Board:

```
esp32:
  board: esp32-s3-devkitc-1
  flash_size: 16MB
  framework:
    type: esp-idf
psram:
  mode: octal
  speed: 80MHz
light:
  - platform: esp32_rmt_led_strip
    name: "RGB-LED"
    pin: GPIO48
    num_leds: 1
    rgb_order: GRB
    chipset: ws2812
```

Dazu kommen *i2s_audio*, *microphone* (Pins wie in Beispiel 6), *micro_wake_word* und *voice_assistant* – Anleitung auf `esphome.io`.

Weitere Wege

ESP-IDF + ESP-DL

Espressifs eigene KI-Bibliothek mit fertigen Modellen und voller Nutzung der Vektorbefehle. Für Fortgeschrittene, die von der Arduino IDE auf ESP-IDF umsteigen.

Kamera-KI

Personen- und Gesichtserkennung (ESP-WHO) brauchen eine Kamera – dafür gibt es eigene ESP32-S3-Kameraboards.

Cloud-KI

Der S3 kann per HTTPS einen Cloud-Dienst mit großem Sprachmodell abfragen. Dann rechnet nicht der Chip, sondern der Anbieter; du brauchst einen API-Schlüssel, und die Daten verlassen dein Netz.

09 Wenn es nicht klappt

SYMPTOM	LÖSUNG
„PSRAM: 0 MB“ oder Neustart-Schleife mit „PSRAM ID read error“	`Werkzeuge → PSRAM → OPI PSRAM` wählen. Das N16R8 hat Octal-PSRAM – „QSPI PSRAM“ passt nicht.
Kein Port in der IDE	Kabel an die Buchse **COM**. Anderes USB-C-Kabel probieren. Windows 10/11 erkennt den Wandler meist von selbst, sonst den passenden Treiber des Chipherstellers WCH von `wch-ic.com` installieren.
Upload hängt bei „Connecting...“	**BOOT** gedrückt halten, kurz **RST** drücken, BOOT loslassen – dann erneut hochladen.
Monitor bleibt leer	115200 Baud? Kabel an **COM**? Bei „USB CDC On Boot: Enabled“ geht die Ausgabe an die andere Buchse.
Nach Beispiel 3 kein Upload mehr über „USB“	Normal – die Buchse ist jetzt eine Tastatur. Über **COM** hochladen oder BOOT + RST drücken.
GPIO35, 36 oder 37 reagiert nicht	Diese Pins sind beim N16R8 vom PSRAM belegt – anderen Pin nehmen.
„Sketch too big“	Partition Scheme mit 3 MB APP wählen (siehe Seite 5).
Bluetooth-Seriell (`BluetoothSerial`) fehlt	Der S3 kann nur Bluetooth LE, kein klassisches Bluetooth. Die BLE-Beispiele im Board-Paket nutzen.
Ständige Neustarts unter Last	„Brownout“ – der USB-Port liefert zu wenig Strom. Anderer Port, aktiver Hub oder kürzeres Kabel.

Passt dazu

INMP441 Mikrofon
Für Sprachbefehle & Audio-KI
`MKR-INMP441-64`

GY-521 MPU-6050
Bewegungsdaten für Gesten-KI
`MKR-GY-521-017`

WS2812 LED-Ring
Status der KI anzeigen
`MKR-WS2812-RING12-67`

SHT40 Temp. & Feuchte
Präzise Sensordaten
`MKR-SHT40-76`

Sicherheitshinweise. Elektronikmodul für Ausbildungs-, Entwicklungs- und Experimentierzwecke. Nur mit Kleinspannung betreiben, niemals an Netzspannung. Der ESP32-S3 wird bei rechenintensiven Anwendungen spürbar warm – nicht abdecken. Nicht geeignet für Kinder unter 14 Jahren. Elektroaltgeräte gehören nicht in den Hausmüll – Rückgabe kostenlos bei kommunalen Sammelstellen.

Verantwortlicher Wirtschaftsakteur (GPSR): Alexander Prez · makeroo · Hinterm Bahnhof 4 · 90513 Zirndorf · support@makeroo.de
Fragen? Schreib uns mit einem Foto deines Aufbaus.
Arduino, ESP32, ESPHome, TensorFlow und Edge Impulse sind Marken ihrer Inhaber und dienen nur der Kompatibilitätsangabe.