# AviFeeder – Informe sobre IA

Alejo Burnowicz, Juan Pablo Correa, Agustin Federico, Gonzalo Matellan, Nahuel Repetto

42646860, 40653000, 41882938, 43325268, 42024337
Martes, Grupo 4

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

**Resumen.**

Este informe presenta en detalle el proceso de desarrollo y evaluación del código embebido de nuestro proyecto utilizando la metodología de VibeCoding total. A lo largo del documento se describen las herramientas empleadas, el modelo de inteligencia artificial seleccionado y los prompts utilizados para guiar la generación automática del código. También se presenta el tiempo requerido para lograr la funcionalidad completa mediante VibeCoding. Se incluyen capturas de pantalla de las métricas de rendimiento (uso de CPU y memoria) obtenidas en ambos enfoques.

## 1 Herramienta y Modelo

Para el desarrollo del trabajo se utilizó Claude AI (modelo Sonnet 4.5), aprovechando su capacidad de razonamiento extendido para mejorar la planificación y coherencia del código generado. Esta herramienta de inteligencia artificial asistió en la redacción, estructuración y depuración del programa para el ESP32, contribuyendo a una implementación organizada y alineada con los requisitos funcionales del sistema embebido.

## 2 Prompt

La prompt utilizada corresponde al tipo Prompt Pattern, presenta una plantilla estructurada con campos específicos que describen el lenguaje, la plataforma, el objetivo, los requisitos funcionales del sistema y detalles de compatibilidad y configuración, lo que permite al modelo generar un código coherente y completo a partir de dicha información.

A continuación se presenta la prompt final refinada:
LENGUAJE:
- Wiring/Arduino (ESP32).

PLATAFORMA:
- ESP32 (usar FreeRTOS, tareas y colas).

OBJETIVO:
- Generar el firmware para un alimentador automático que lea sensores (ultrasonido y potenciómetros) y controle un servo, dos LEDs, conectividad Wi-Fi y envío de logs por MQTT en formato JSON. Debe respetar estrictamente la arquitectura indicada a continuación.

REQUISITOS FUNCIONALES:

1. Sensores:
    - Sensor ultrasónico (trigger + echo) para detectar objeto cercano.
    - Un potenciómetro analógico para detectar nivel de agua.
    - Un segundo potenciómetro analógico para medir peso.
2. Actuadores:
    - Un servo que abra/cierre el dispensador.
    - Dos LEDs (uno para agua y otro para comida) controlados con PWM (ledc).
3. Conectividad:
    - Conectar a Wi-Fi y publicar logs periódicos por MQTT (JSON).
    - Suscribirse a un topic para recibir comandos que puedan encender/apagar LEDs o desencadenar un “shaker” (parpadeo).
4. Lógica:
    - Implementar una máquina de estados con al menos 3 estados, uno de ellos puede ser estado de inicio. Debe usarse un `switch` por estado y dentro de cada estado otro `switch` por evento (patrón de switches anidados).
    - Implementar función `getevent()` que únicamente lea sensores y setee eventos. No leer sensores en otros lugares.
    - Los cambios de estado solo deben ocurrir dentro del `switch` central de la máquina de estados.
    - Usar FreeRTOS: al menos una tarea concurrente para mover el servo y otra tarea para conectividad/MQTT/envío de logs.
    - Utilizar temporizadores para periodicidad de logs y lectura de sensores.
    - Cuando el valor indicado por el potenciómetro que indica peso sea menor que un umbral se debe mover el servo para dispensar comida; cuando supere ese mismo umbral, volver a posición.

- Si el otro potenciómetro indica bajo nivel de agua, encender LED de agua; si el ultrasonido indica objeto lejano, encender LED de comida.
- Soportar recepción de mensajes MQTT con comandos predefinidos para: encender/apagar LED de agua, encender/apagar LED de comida, y ejecutar “LED_SHAKER” (parpadeo de ambos LEDs).

5. Comunicación/Logs:
   - Cada segundo publicar por MQTT un JSON con al menos: tiempo (s), weight (g), potValue, distance (cm), estado actual, evento actual.
   - Serial logs para debugging.

DETALLES DE IMPLEMENTACIÓN Y COMPATIBILIDAD:

- NO USAR ledcSetup() y ledcAttachPin() para controlar los LEDs PWM, en su lugar debe emplearse la API actual ledcAttachChannel() conforme a los cambios introducidos en la migración del core 2.x a 3.0.
- Evitar errores de crosses initialization y jump to case label en los switch de la máquina de estados.
- Usar canales LEDC altos (por ejemplo 6 y 7) para los LEDs para evitar conflictos con el servo.
- No usar números mágicos.
- Todo el código debe estar en inglés.

CONFIGURACIÓN:

- Librerías mínimas: <WiFi.h>, <PubSubClient.h>, <ESP32Servo.h>, <freertos/FreeRTOS.h>, <freertos/task.h> y "ArduinoJson.h".
- Definiciones de pines: pin led de agua (22), pin led de comida (23), pin trigger (19), pin echo (18), pin servo (5), pin potenciómetro agua (34) y pin potenciómetro peso(35).
- Definiciones de umbrales: umbral de potenciometro de peso bajo (1000), umbral de distancia (20), umbral de potenciometro de agua (500), posición normal del servo (120), posición abierta del servo (135).
- Wi-Fi: Utilizar ssid "Wokwi-GUEST".
- MQTT: servidor MQTT "broker.hivemq.com" y tópico "AviFeeder/Datos".

## 3 Tiempo

El proceso completo de conseguir un firmware funcional demandó aproximadamente cinco horas desde la redacción del primer prompt, durante las cuales se realizaron múltiples iteraciones para refinar el código generado por Claude Sonnet 4.5. En una de las iteraciones iniciales surgieron errores de compilación como *crosses initialization* y *jump to case label*, que impidieron la correcta estructuración del flujo del programa. En una iteración posterior, el código falló nuevamente debido al uso incorrecto del parámetro *channel* en lugar del pin al ejecutar *ledcWrite*, además de presentarse conflictos con los canales *LEDC* asignados. Después de depurar estos

problemas y ajustar el diseño general, se logró obtener un firmware estable y acorde a los requisitos del sistema.

## 4 Métricas

CASO DE PRUEBA 1: Ejecución 10 segundos sin hacer interacción. En ambos casos initStats() se ejecuto al comienzo de setup() para obtener resultados comparables, ya que el sistema con *vibecoding* se conecta a Wi-Fi dentro del setup, mientras que el sistema sin *vibecoding* lo hace luego del setup.

Sistema embebido con *vibecoding*:

```
=== Contribución Promedio al total del sistema ===
=== Tiempo de muestreado:     10(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  95.38% | Ocupado:  18.35% | Libre (IDLE):  77.04%
Core 1 -> Total: 107.45% | Ocupado:  17.56% | Libre (IDLE):  89.89%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 333160 bytes
Heap libre : 231516 bytes
Heap usado : 101643 bytes
Uso        : 30.51 %

Muestreo de Metricas finalizado
```

Sistema embebido sin *vibecoding*:

```
=== Contribucion Promedio al total del sistema ===
=== Tiempo de muestreado:     10(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  93.09% | Ocupado:  25.01% | Libre (IDLE):  68.08%
Core 1 -> Total: 110.67% | Ocupado:  28.57% | Libre (IDLE):  82.10%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 333064 bytes
Heap libre : 236761 bytes
Heap usado : 96302 bytes
Uso        : 28.91 %

Muestreo de Metricas finalizado
```

CASO DE PRUEBA 2: Ejecución hasta apagado de LED por suficiente agua. Ambos sistemas embebidos siguieron los mismos eventos, Cierre del servo por peso suficiente → Encendido de LED por falta de comida → Apertura del servo por peso insuficiente → Apagado de LED por suficiente agua.

Sistema embebido con *vibecoding*:

```
=== Contribución Promedio al total del sistema ===
=== Tiempo de muestreado:     14(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  96.66% | Ocupado:  16.24% | Libre (IDLE):  80.42%
Core 1 -> Total: 106.46% | Ocupado:  17.68% | Libre (IDLE):  88.78%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 333160 bytes
Heap libre : 226889 bytes
Heap usado : 106270 bytes
Uso        : 31.90 %

Muestreo de Metricas finalizado
```

Sistema embebido sin *vibecoding*:

```
=== Contribución Promedio al total del sistema ===
=== Tiempo de muestreado:     14(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  95.38% | Ocupado:  21.58% | Libre (IDLE):  73.80%
Core 1 -> Total: 108.81% | Ocupado:  36.48% | Libre (IDLE):  72.32%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 333072 bytes
Heap libre : 233534 bytes
Heap usado : 99537 bytes
Uso        : 29.88 %

Muestreo de Metricas finalizado
Connected
```

## 5 Rúbricas

RUBRICA FUNCIONALIDAD EMBEBIDO

| CRITERIOS | SUFICIENTEMENTE LOGRADO (A) | MEDIANAMENTE LOGRADO (B) | INSUFICIENTEMENTE LOGRADO (C) | Puntaje | Eval |
|---|---|---|---|---|---|
| **FUNCIONA SIN ERRORES** | -funciona el 100% de las veces sin errores | -funciona el 70% de las veces sin errores | Funciona el 50% de las veces sin errores | A: Hasta 20 pts.<br>B: Hasta 15 pts.<br>C: 0 | 20 |
| **EMPLEA TAREAS DE FREERTOS** | -utiliza varias tareas de freertos | -Utiliza una sola tarea con freertos | -no utiliza freertos | A: Hasta 20 pts.<br>B: Hasta 10 pts<br>C: 0 | 20 |
| **NO USA ESPERAS BLOQUEANTES** | Reemplaza las espera bloqueantes por tareas con freertos<br>Utiliza temporizadores con freertos | -utiliza temporizadores sin freertos | -utiliza espera bloqueantes | A: Hasta 20 pts.<br>B: Hasta 15 pts.<br>C: 0 | 0 |
| **UTILIZA LOS SENSORES Y ATUADORES SOLICITADOS** | -utiliza la cantidad de sensores y actuadores solicitados de manera eficiente | -utiliza la cantidad de sensores y actuadores de manera poco efectiva | -no utiliza la cantidad de sensores y actuadores solicitados o no funcionan de manera correcta. | A: Hasta 20 pts.<br>B: Hasta 12 pts.<br>C: 0 | 20 |
| **SE COMUNICA CON BORKER CON MQTT** | -funciona correctamente el sistema usando la comunicación por mqtt sin problemas | -funciona el sistema correctamente, pero posee algunos problemas de funcionamiento el sistema. | No funciona la comunicación mqtt.<br>Funciona la comunicación, pero falla en el funcionamiento del sistema | A: Hasta 20 pts.<br>B: Hasta 15 pts.<br>C: 0 | 20 |
| **TOTAL** | | | | **100** | 80 |

RUBRICA LEGIBILIDAD Y MANTENIBILIDAD EMBEBIDO

| CRITERIOS | SUFICIENTEMENTE LOGRADO (A) | MEDIANAMENTE LOGRADO (B) | INSUFICIENTEMENTE LOGRADO (C) | Puntaje | Eval |
|---|---|---|---|---|---|
| **PATRON DE MAQUINA DE ESTADOS** | -respeta un 90% el patron de maquina de estados | -respeta un 70% el patron de maquina de estados | Respeta el patron de maquina de estados menos de un 70% | A: Hasta 20 pts.<br>B: Hasta 15 pts.<br>C: 0 | 20 |
| **TAMAÑO DE FUNCIONES** | Funciones < a 30 lineas | 30 lineas< Funciones < 50 lineas | Funciones >50 lineas | A: Hasta 20 pts.<br>B: Hasta 10 pts<br>C: 0 | 5 |
| **NOMBRES DESCRIPTIVOS VARIABLES Y METODOS** | Utiliza nombres descriptivos usando convención | Utiliza nombres descriptivos pero sin usar convención | Los nombres no son descriptivos | A: Hasta 15 pts.<br>B: Hasta 10 pts.<br>C: 0 | 15 |
| **COMENTARIOS EN CODIGO** | -posee el 60% del código con comentarios explicando que hace cada parte el mismo | -posee el 20% del código con comentarios explicando cada parte del código | -posee menos del 20% del código con comentarios explicando cada parte del código | A: Hasta 10 pts.<br>B: Hasta 5 pts.<br>C: 0 | 10 |
| **CODIGO MODULAR** | - el código esta separado en módulos y no se encuentra acoplado | El código esta separado en módulos, pero se encuentra medianamente acoplado | El código no es modular y esta fuertemente acoplado | A: Hasta 20 pts.<br>B: Hasta 15 pts.<br>C: 0 | 15 |
| **CANTIDAD DE WARNING ANDROID** | Tiene menos de 10 warning | 10<Tiene entre<20 | Tiene mas de 20 warning | No aplica | - |
| **TOTAL** | | | | **85** | 65 |

# 6 Referencias

1. Espressif Systems. Migración de la versión 2.x a 3.0 – Núcleo Arduino ESP32. [En línea]. Disponible en: https://docs.espressif.com/projects/arduino-esp32/en/latest/migration_guides/2.x_to_3.0.html . Consultado: 13 noviembre 2025.