

Arquitectura de Alto Rendimiento para Sistemas Distribuidos en el Borde: Evaluación Técnica de Plataformas ESP32 y RISC-V para la Implementación de APU_filter y la Lógica flux_condenser.py

1. Definición del Sistema y Paradigma Arquitectónico

La convergencia de la teoría del caos, el análisis topológico de datos (TDA) y la inteligencia artificial distribuida está redefiniendo los límites de lo que es posible en la computación en el borde (Edge Computing). El presente informe técnico aborda la viabilidad, el diseño y la implementación del sistema distribuido denominado **APU_filter**, un ecosistema de nodos de sensores y procesamiento interconectados, y su componente lógico central, **flux_condenser.py**. Este componente de software no es un simple script de control, sino el núcleo algorítmico encargado de gestionar dinámicas no lineales, procesar señales en estructuras de grafos y "condensar" flujos de datos de alta entropía en información topológica procesable.

La motivación subyacente para esta investigación radica en las limitaciones inherentes de las arquitecturas centralizadas en la nube para manejar fenómenos físicos de alta velocidad y baja latencia. Cuando se trata de sistemas que exhiben comportamientos caóticos —como los modelados por los atractores de Lorenz— o que requieren la detección de anomalías en tiempo real basada en la forma de los datos (TDA), la latencia de ida y vuelta a un servidor remoto es inaceptable. El procesamiento debe ocurrir *in situ*, en el "borde extremo". Esto plantea un desafío monumental de ingeniería: ¿cómo ejecutar cargas de trabajo matemáticas propias de estaciones de trabajo científicas en microcontroladores con restricciones térmicas y energéticas severas?

La respuesta reside en la selección meticulosa de una arquitectura de hardware heterogénea y la adopción de patrones de diseño de software avanzados, como el **Data Mesh** (Malla de Datos). A diferencia de los enfoques monolíticos tradicionales donde los microcontroladores actúan como simples tuberías de datos crudos hacia un lago de datos central, el paradigma de Data Mesh propone que cada nodo (o dominio de nodos) sea propietario de sus datos y los sirva como un "producto" refinado. En el contexto de APU_filter, esto significa que un nodo basado en el **ESP32-P4** no transmite lecturas de acelerómetro a 1 kHz; en su lugar, ejecuta **flux_condenser.py** para analizar la topología de la señal, calcular su dimensión fractal y

transmitir únicamente la detección de un cambio de fase o una anomalía estructural.¹ Esta investigación evalúa exhaustivamente el ecosistema de hardware necesario para soportar esta visión, contrastando la madurez del **ESP32-S3** y la potencia bruta de la nueva serie **ESP32-P4** de Espressif Systems, frente a competidores emergentes como el **Raspberry Pi RP2350** y las plataformas de alto rendimiento basadas en **Cortex-M7**. Se analizará cómo las extensiones vectoriales, la gestión de memoria PSRAM y los coprocesadores de comunicación dedicados son factores determinantes para la ejecución eficiente de algoritmos de caos y TDA.

2. Análisis Profundo de Plataformas de Hardware: La Frontera entre MCU y MPU

La selección del silicio para el nodo central de procesamiento de APU_filter es la decisión más crítica del proyecto. El hardware no solo debe ejecutar el código, sino que debe hacerlo con un determinismo temporal absoluto para garantizar la estabilidad de los algoritmos de control caótico.

2.1. ESP32-P4: La Nueva Vanguardia de Espressif

El **ESP32-P4** representa un cambio filosófico y arquitectónico significativo por parte de Espressif Systems. Históricamente conocida por integrar conectividad Wi-Fi y Bluetooth en cada chip, con el P4, Espressif ha optado por disociar la radio del núcleo de procesamiento principal. Esta decisión, lejos de ser una desventaja, posiciona al P4 como un candidato superior para sistemas de control crítico y procesamiento de datos intensivo como APU_filter.³

2.1.1. Arquitectura del Núcleo RISC-V y Potencia de Cómputo

El corazón del ESP32-P4 es un procesador de doble núcleo basado en la arquitectura **RISC-V** que opera a una frecuencia de hasta **400 MHz**. Este incremento de frecuencia, comparado con los 240 MHz típicos de la serie ESP32 anterior, ofrece un margen de maniobra esencial para la integración numérica de ecuaciones diferenciales no lineales, como las requeridas para simular el atractor de Lorenz en `flux_condenser.py`.³

La arquitectura RISC-V del P4 incluye extensiones personalizadas para acelerar operaciones de Inteligencia Artificial (IA) y Procesamiento Digital de Señales (DSP). Estas extensiones permiten la ejecución eficiente de operaciones matriciales y convoluciones, fundamentales para las redes neuronales de grafos (GNN) y los algoritmos de filtrado avanzado. Además, el P4 integra una Unidad de Punto Flotante (FPU) de precisión simple, lo cual es vital para el cálculo de distancias euclidianas en algoritmos de TDA como la filtración de Vietoris-Rips sin incurrir en la penalización masiva del software de emulación de punto flotante.³

Un aspecto crucial para la eficiencia energética en APU_filter es la arquitectura asimétrica del P4, que incluye un núcleo de baja potencia (**LP-Core**) capaz de operar hasta 40 MHz. Este núcleo puede encargarse de la monitorización básica de sensores y la gestión de la energía mientras los núcleos de alto rendimiento (HP-Cores) permanecen en sueño profundo, despertando solo cuando la lógica `flux_condenser.py` detecta una condición que requiere

procesamiento intensivo.³

2.1.2. Jerarquía de Memoria y Manejo de Grandes Datos

Los algoritmos de Análisis Topológico de Datos (TDA) y el procesamiento de grafos son notoriamente voraces en cuanto a memoria. La construcción de un complejo simplicial o el almacenamiento de una matriz de adyacencia para un grafo dinámico puede desbordar rápidamente la SRAM interna de un microcontrolador estándar (típicamente 512 KB).

El ESP32-P4 aborda este cuello de botella soportando hasta **32 MB de PSRAM** (Pseudo-Static RAM) externa, accesible a través de un bus de alta velocidad y caché de dos niveles. Esta capacidad es transformadora para APU_filter. Permite mantener en memoria "ventanas" de datos históricos mucho más amplias para el análisis de series temporales, o cargar modelos de TinyML más complejos y precisos que no necesitan ser cuantizados tan agresivamente como en plataformas menores.⁴ La gestión eficiente de esta memoria, utilizando estructuras de datos dispersas (sparse matrices), será clave para la implementación de algoritmos de grafos en `flux_condenser.py`, evitando la saturación del bus de memoria.⁶

2.1.3. Desacoplamiento de la Radio y Estabilidad Temporal

Para aplicaciones de control en tiempo real, la "fluctuación" (jitter) introducida por las interrupciones de la radio Wi-Fi es un problema clásico. En el ESP32-S3 y modelos anteriores, los núcleos de la CPU deben atender las demandas de tiempo crítico de la pila Wi-Fi, lo que puede pausar la ejecución del código de usuario.

El diseño del ESP32-P4 requiere un chip compañero para la conectividad inalámbrica, típicamente el **ESP32-C6** (que soporta Wi-Fi 6, Bluetooth 5 y Thread/Matter).⁷ Esta arquitectura de dos chips es ideal para APU_filter. El ESP32-C6 actúa como un procesador de comunicaciones dedicado (Network Co-processor), gestionando toda la complejidad de la capa física, la encriptación y el mantenimiento de la conexión. El ESP32-P4 se comunica con el C6 a través de una interfaz SDIO o SPI de alta velocidad. Esto garantiza que la ejecución de `flux_condenser.py` en el P4 sea determinista y no se vea interrumpida por la negociación de claves Wi-Fi o retransmisiones de paquetes, un requisito indispensable para mantener la coherencia en sistemas caóticos sincronizados.⁸

2.1.4. Interfaces HMI y Multimedia

Si APU_filter requiere visualización local de los atractores o del estado del grafo, el P4 sobresale gracias a su soporte nativo para **MIPI-DSI** (Display Serial Interface) y **MIPI-CSI** (Camera Serial Interface). Esto permite conectar pantallas de alta resolución y cámaras directamente al chip, con aceleración hardware para codificación H.264 y procesamiento de píxeles (PPA), liberando a la CPU para tareas lógicas.³ La fluidez en la interfaz de usuario (UI), lograda gracias a la descarga del trabajo gráfico, es un diferenciador clave respecto a soluciones basadas en SPI tradicionales.¹¹

2.2. ESP32-S3: El Equilibrio Maduro para Nodos Sensores

Mientras que el P4 es el cerebro central ideal, el **ESP32-S3** se perfila como la opción óptima para los nodos sensores distribuidos dentro de la red de APU_filter. Basado en un doble núcleo **Xtensa LX7** a 240 MHz, el S3 incluye instrucciones vectoriales específicas que aceleran las bibliotecas ESP-NN y ESP-DSP.¹²

La integración nativa de Wi-Fi y Bluetooth LE en el S3 simplifica el diseño del hardware de los nodos periféricos, reduciendo el costo y el tamaño de la PCB. Aunque su potencia de cómputo es menor que la del P4, es suficiente para realizar pre-procesamiento de señales (FFT, filtrado digital) y ejecutar inferencias ligeras de TinyML antes de enviar los datos al nodo central vía ESP-NOW. Su madurez en el ecosistema de software asegura una implementación rápida y fiable de los drivers de sensores.¹⁴

2.3. Análisis de Alternativas: RP2350 y Cortex-M7

Para validar la elección de la plataforma ESP32, es necesario examinar las alternativas más competitivas en el mercado actual.

2.3.1. Raspberry Pi RP2350 (Pico 2)

El **RP2350**, lanzado recientemente, introduce una arquitectura híbrida fascinante que permite seleccionar en el arranque entre núcleos ARM Cortex-M33 o núcleos RISC-V Hazard3, ambos a 150 MHz.¹⁶

La característica más disruptiva del RP2350 para APU_filter es su bloque de E/S Programable (**PIO**). Con 12 máquinas de estado PIO, el RP2350 puede implementar protocolos de comunicación personalizados o interfaces de sensores exóticos con una precisión de ciclo de reloj, sin carga para la CPU principal. Esto lo hace insuperable para la adquisición de datos de latencia ultra-baja o para controlar actuadores de alta velocidad.¹⁸

Además, el RP2350 incorpora características de seguridad robustas como **TrustZone** para Cortex-M y arranque firmado, lo que es esencial si APU_filter opera en un entorno hostil donde la integridad del firmware debe ser garantizada criptográficamente.¹⁸

Sin embargo, su limitación principal es la potencia de cálculo bruta. A 150 MHz, es significativamente más lento que el ESP32-P4 para tareas de cálculo numérico pesado.

Además, la versión inalámbrica (Pico 2 W) ha experimentado retrasos en la madurez de su soporte de software (especialmente en MicroPython y drivers de red estables), lo que representa un riesgo para despliegues inmediatos en 2026.²⁰

2.3.2. Teensy 4.1 (Cortex-M7)

El **Teensy 4.1**, basado en el NXP i.MX RT1062, es la referencia en rendimiento puro para microcontroladores, con un núcleo ARM Cortex-M7 corriendo a **600 MHz**. Si la lógica `flux_condenser.py` requiere procesamiento de audio en tiempo real, síntesis de señales complejas o bucles de control PID de frecuencia extremadamente alta (> 100 kHz), el Teensy es la plataforma superior.¹²

Su unidad de punto flotante (FPU) de doble precisión es una ventaja significativa para cálculos científicos que requieren alta estabilidad numérica, como simulaciones de caos a largo plazo. Sin embargo, su talón de Aquiles es la falta de conectividad inalámbrica

integrada. Añadir Wi-Fi o Bluetooth requiere módulos externos, lo que complica la integración y la pila de software en comparación con la solución nativa de Espressif.¹²

2.4. Síntesis y Recomendación de Hardware

La evidencia sugiere una arquitectura jerárquica para APU_filter:

1. **Nodo Central (Gateway/Procesador): ESP32-P4** acoplado con **ESP32-C6**. Esta combinación ofrece el máximo rendimiento computacional (400 MHz RISC-V + Extensiones AI), gran memoria (32 MB PSRAM) para TDA y grafos, y una conectividad robusta y desacoplada. Es la plataforma ideal para alojar la lógica pesada de flux_condenser.py.³
2. **Nodos Sensores (Satélites): ESP32-S3**. Su equilibrio entre potencia (instrucciones vectoriales para pre-procesamiento), conectividad integrada y costo lo hace ideal para la recolección distribuida de datos.¹³
3. **Controladores de Actuadores Especializados (Opcional): RP2350**. Si el sistema requiere interfaces de hardware no estándar o latencia de E/S crítica, el RP2350 puede integrarse como un subsistema de E/S inteligente, comunicado con el P4 vía SPI o UART.¹⁸

3. La Lógica flux_condenser.py: Dinámica, Caos y Topología

El nombre "flux_condenser.py" evoca la gestión de flujos dinámicos y la condensación de información. Desde una perspectiva técnica rigurosa, esto se traduce en la implementación de tres pilares algorítmicos: Dinámica de Sistemas Caóticos (Atractores de Lorenz), Análisis Topológico de Datos (TDA) y Redes Neuronales de Grafos (GNN).

3.1. Dinámica Caótica y Atractores de Lorenz en el Borde

El sistema de Lorenz, definido por las ecuaciones diferenciales acopladas $\frac{dx}{dt} = \sigma(y - x)$, $\frac{dy}{dt} = x(\rho - z) - y$, y $\frac{dz}{dt} = xy - \beta z$, es el paradigma del comportamiento caótico determinista. Para parámetros canónicos ($\sigma=10$, $\beta=8/3$, $\rho=28$), el sistema exhibe el famoso atractor de mariposa, donde trayectorias cercanas divergen exponencialmente.²²

3.1.1. Aplicaciones en Control y Seguridad

La implementación de este atractor en flux_condenser.py no es meramente académica. Tiene aplicaciones industriales directas:

- **Modulación de Controladores PID:** La señal caótica del atractor puede utilizarse para modular dinámicamente las ganancias (K_p , K_i , K_d) de un controlador PID. Esto dota al sistema de control de una capacidad de exploración del espacio de estados, permitiéndole escapar de mínimos locales o evitar resonancias mecánicas fijas en sistemas de vibración industrial. Estudios demuestran que la optimización de parámetros PID mediante dinámicas caóticas mejora la robustez ante perturbaciones

no lineales.²⁴

- **Sincronización Segura:** Dos instancias de APU_filter (transmisor y receptor) pueden sincronizar sus atractores de Lorenz internos (sincronización maestro-esclavo). La información útil se "oculta" dentro de la señal portadora caótica, pareciendo ruido térmico para un observador externo, pero siendo perfectamente recuperable por el receptor sincronizado. Esto constituye una capa física de encriptación de bajo costo computacional.²⁷

3.1.2. Desafíos de Implementación Numérica

La simulación precisa de estos sistemas en un microcontrolador requiere métodos de integración numérica robustos, como **Runge-Kutta de 4to orden (RK4)**. El método de Euler, aunque más simple, introduce errores de truncamiento que destruyen la estructura fina del atractor a largo plazo.²⁸

En un entorno de Python embebido (MicroPython o CircuitPython), la implementación ingenua de RK4 utilizando bucles y listas nativas es prohibitivamente lenta. La solución crítica es el uso de la biblioteca **ulab**. ulab es una reimplementación de NumPy escrita en C optimizado para microcontroladores. Permite realizar operaciones vectoriales (productos punto, sumas de arrays, broadcasting) a velocidades cercanas al código C nativo, eliminando la sobrecarga del intérprete de Python.³⁰

- *Implementación Recomendada:* Definir el estado del sistema como un `ulab.numpy.array` y calcular los incrementos de RK4 mediante operaciones vectorizadas. Esto permite que `flux_condenser.py` actualice el estado del atractor en tiempo real (ej. > 100 Hz) en un ESP32-P4.³²

3.2. Análisis Topológico de Datos (TDA): La "Condensación" de la Forma

El TDA ofrece una metodología para analizar la "forma" de los datos de alta dimensión, siendo robusto al ruido y a las deformaciones. La herramienta principal es la **Homología Persistente**, que identifica características topológicas (componentes conectados, bucles, cavidades) que persisten a través de diferentes escalas espaciales.³³

3.2.1. Filtración de Vietoris-Rips

El proceso estándar implica construir un complejo simplicial de Vietoris-Rips a partir de una nube de puntos (lecturas de sensores multivariados). A medida que se incrementa un parámetro de umbral $\$\\epsilon$, se conectan puntos y se forman simplices. El seguimiento de cuándo nacen y mueren los agujeros en este complejo genera un "código de barras" de persistencia o un diagrama de persistencia.³⁵

El desafío computacional es inmenso: el número de simplices puede crecer exponencialmente con el número de puntos. En un microcontrolador con RAM limitada, esto es el factor limitante. El ESP32-P4, con sus 32 MB de PSRAM, es una de las pocas plataformas embebidas capaces de manejar complejos de Rips de tamaño moderado sin recurrir a almacenamiento externo lento.⁵

3.2.2. Algoritmos de Streaming para el Borde

Dado que APU_filter procesa datos continuos (streaming), no es viable recalcular la homología desde cero en cada paso. flux_condenser.py debe implementar estrategias de TDA para streaming:

- **Ventanas Deslizantes:** Mantener una nube de puntos que representa solo los últimos N eventos.
- **Sketching y Submuestreo:** Utilizar algoritmos como "Witness Complex" o submuestreo basado en "Maxmin" (farthest point sampling) para seleccionar un subconjunto representativo de puntos "landmark" sobre los cuales construir el complejo, reduciendo drásticamente la carga computacional sin sacrificar la topología global.³⁸
- **Librerías Híbridas:** No existe una librería TDA completa en MicroPython puro. La estrategia recomendada es portar el núcleo computacional de librerías ligeras de C++ como **Ripser** o módulos selectos de **GUDHI** como extensiones C nativas para MicroPython (archivos .mpy o integrados en el firmware). Esto delega el trabajo pesado al hardware (y a la FPU del P4) mientras la lógica de alto nivel permanece en Python.⁴⁰

3.3. Grafos y Detección de Anomalías (TinyDL y GNN)

El sistema APU_filter puede conceptualizarse como un grafo dinámico donde los nodos son sensores y las aristas representan correlaciones temporales o espaciales. Una anomalía en el sistema (ej. fallo estructural) se manifiesta no solo como un valor fuera de rango, sino como una ruptura en la estructura del grafo (ej. dos sensores que siempre vibran en fase dejan de hacerlo).⁴³

3.3.1. Detección Out-of-Distribution (OOD)

Las Redes Neuronales de Grafos (GNN) son excepcionales para capturar estas dependencias. Para flux_condenser.py, se propone el uso de modelos GNN ligeros entrenados para predecir la estructura futura del grafo. Un error alto en la predicción indica una anomalía "Out-of-Distribution" (OOD), es decir, un comportamiento nunca antes visto.⁴⁵

3.3.2. Implementación con Matrices Dispersas

La representación eficiente de grafos en memoria es crítica. Almacenar una matriz de adyacencia densa para N nodos consume $O(N^2)$ memoria. Es imperativo utilizar formatos de matrices dispersas como CSR (Compressed Sparse Row) o COO (Coordinate List).

Aunque scipy.sparse no está disponible directamente en MicroPython, se pueden implementar estructuras CSR utilizando arrays compactos de ulab (un array para valores, otro para índices de columna, otro para punteros de fila). Esto permite realizar la operación fundamental de las GNN —la multiplicación matriz-vector dispersa (SpMV)— de manera eficiente en el ESP32-P4, aprovechando sus capacidades vectoriales.⁶

4. Infraestructura de Comunicación Distribuida: La

Malla en el Borde

La inteligencia de `flux_condenser.py` depende de la calidad y latencia de los datos que recibe. La infraestructura de comunicación de `APU_filter` debe ser robusta, de baja latencia y escalable.

4.1. ESP-NOW: El Protocolo para Tiempo Real

Para la comunicación intra-sistema (entre nodos sensores y el nodo central P4), **ESP-NOW** es la elección técnica superior frente a Wi-Fi estándar o Bluetooth Mesh. ESP-NOW es un protocolo "sin conexión" que utiliza tramas de acción de proveedor (Vendor Specific Action Frames) sobre la capa física IEEE 802.11.⁴⁹

- **Latencia Determinista:** Al eliminar la necesidad de asociación a un Punto de Acceso (AP), intercambio de claves de 4 vías (4-way handshake) y cabeceras TCP/IP, ESP-NOW reduce la latencia de transmisión a niveles de milisegundos, comparable a protocolos industriales cableados.
- **Capacidad de Carga:** La versión 2.0 de ESP-NOW (disponible en ESP-IDF v5.x) soporta cargas útiles de hasta **1470 bytes**. Esto es crucial para `APU_filter`, ya que permite transmitir no solo lecturas simples, sino vectores de características embebidos, pequeños tensores de TinyML o "firmas" topológicas compactas sin necesidad de fragmentación compleja a nivel de aplicación.⁵¹

4.2. Estrategias de Fiabilidad y Topología

ESP-NOW no garantiza la entrega en la capa de aplicación (similar a UDP). Para construir un sistema fiable sobre él, `flux_condenser.py` debe implementar:

1. **Mecanismo de ACK Lógico:** Aunque la capa MAC confirma la recepción física (`ESP_NOW_SEND_SUCCESS`), se requiere un ACK de aplicación para confirmar que el dato fue procesado o encolado correctamente por el receptor.⁵¹
2. **Control de Flujo:** Es imperativo esperar a que el callback de envío retorne antes de injectar el siguiente paquete para evitar desbordar la cola interna del driver Wi-Fi, lo que causaría pérdida de paquetes silenciosa.⁵¹
3. **Topología Híbrida Estrella-Malla:** Se recomienda una topología donde los nodos sensores (ESP32-S3) se comunican directamente con el nodo central (ESP32-P4/C6) en modo unicast para datos críticos. El modo **Broadcast** (a la dirección FF:FF:FF:FF:FF:FF) debe reservarse para señales de sincronización de tiempo o "heartbeats" globales, teniendo en cuenta que las tramas broadcast no soportan encriptación nativa en el hardware actual.⁵¹

4.3. Sincronización Temporal Distribuida

Para correlacionar eventos detectados por sensores distantes, todos los nodos deben compartir una base de tiempo común. En sistemas caóticos, la desincronización de microsegundos puede invalidar el análisis. Se propone un algoritmo de sincronización ligero basado en el intercambio de timestamps vía broadcast, compensando la deriva del reloj

(clock drift) y el tiempo de vuelo, ajustado periódicamente por el nodo central P4 que puede mantener un reloj maestro de alta precisión (RTC calibrado o GPS externo).⁴³

5. Arquitectura Data Mesh en el Borde (Edge Data Mesh)

La implementación de APU_filter debe trascender la arquitectura tradicional de telemetría y adoptar los principios de **Data Mesh**, adaptándolos a las restricciones del mundo embebido.

5.1. Principios de Data Mesh Adaptados

El concepto de Data Mesh, originado en la arquitectura de datos empresarial, se aplica aquí para descentralizar la propiedad y gestión de los datos.¹

1. **Propiedad de Dominio Descentralizada:** Cada nodo inteligente (o clúster local gestionado por un P4) es un "Dominio". El equipo responsable del subsistema de vibración es dueño de los datos de vibración y de la lógica flux_condenser.py que los procesa. No hay un "monolito" central que intente entender todos los datos de la planta.
2. **Datos como Producto:** El nodo no expone registros de memoria crudos. Expone un "Producto de Datos" bien definido y documentado (ej. un objeto JSON o CBOR accesible vía API REST/CoAP) que contiene información de alto valor (ej. "Nivel de Salud del Rodamiento", "Firma Topológica Actual") junto con metadatos de calidad (SLAs de precisión, frescura del dato).⁵³
3. **Infraestructura de Autoservicio:** La plataforma hardware/software (ESP32-P4 + Middleware de Comunicación) actúa como una infraestructura común que permite a los expertos de dominio desplegar sus algoritmos sin tener que reinventar los drivers de red o seguridad.
4. **Gobernanza Computacional Federada:** Las políticas de seguridad (quién puede acceder a los datos, encriptación) se definen globalmente pero se aplican localmente en cada nodo. El uso de características de hardware como **Secure Boot** y **Flash Encryption** en el ESP32, o **TrustZone** en el RP2350, es la implementación física de esta gobernanza, asegurando que solo código y datos autorizados operen en la malla.⁵⁵

5.2. Observabilidad Distribuida

La observabilidad en Data Mesh no es solo logging. Es la capacidad de interrogar al sistema sobre su estado interno basándose en sus salidas. APU_filter debe exponer métricas de salud del propio proceso de datos (tasa de éxito de inferencia, latencia de procesamiento de flux_condenser.py, estabilidad del atractor de Lorenz) como parte de su producto de datos, permitiendo una monitorización proactiva de la calidad de la información.⁵⁶

6. Ecosistema de Software y Optimización

El éxito de APU_filter depende de un stack de software que combine la facilidad de desarrollo de Python con el rendimiento de C++.

6.1. MicroPython vs. CircuitPython en 2026

Para el desarrollo de `flux_condenser.py`, MicroPython se mantiene como la opción más performante y cercana al hardware ("bare metal"). Su gestión de memoria es más manual pero predecible, y su soporte para interrupciones es superior.¹⁵

Sin embargo, CircuitPython está evolucionando rápidamente. Hacia 2026, se espera un soporte maduro para el ESP32-P4, incluyendo módulos nativos para MIPI-DSI (`mipidsi`) que facilitan enormemente la creación de interfaces gráficas sin bloquear el núcleo principal.⁵⁹

- **Recomendación:** Utilizar MicroPython (o un fork optimizado) para el núcleo lógico de control y procesamiento, aprovechando `ulab`. Utilizar CircuitPython si la aplicación requiere una interfaz gráfica rica y rápida iteración en el nodo central P4, aceptando una ligera penalización en rendimiento computacional puro.

6.2. Librerías Clave y Extensiones

- **ulab:** Indispensable. Es el "NumPy para microcontroladores". Cualquier operación matemática en `flux_condenser.py` (FFT, convolución, producto punto) debe usar `ulab` para ser viable.³⁰
- **ESP-DSP / ESP-NN:** Estas librerías de Espressif, escritas en C y ensamblador optimizado, deben ser invocadas desde Python para tareas críticas como la aceleración de redes neuronales o filtros digitales complejos.¹³
- **Implementación de Grafos:** No existen librerías de grafos completas como NetworkX para MicroPython debido a su consumo de memoria. Se recomienda implementar una versión mínima y optimizada que use estructuras de datos dispersas (arrays de `ulab`) para representar solo la topología necesaria, evitando la sobrecarga de objetos Python por cada nodo/arista.⁶²

7. Hoja de Ruta de Implementación y Conclusiones

La investigación concluye que la implementación de **APU_filter** y la lógica **flux_condenser.py** es técnicamente viable y altamente prometedora utilizando la generación de hardware disponible en 2025-2026.

7.1. Recomendación Arquitectónica Final

Se propone una arquitectura **Host-Controller asimétrica**:

- **Núcleo de Cómputo: ESP32-P4** (RISC-V 400MHz, 32MB PSRAM). Ejecuta `flux_condenser.py`, gestiona los atractores de Lorenz, procesa TDA y mantiene el grafo dinámico del sistema.
- **Coprocesador de Red: ESP32-C6**. Gestiona la malla ESP-NOW, la conexión Wi-Fi 6 / Thread y la seguridad de la capa de enlace, garantizando que el P4 opere de manera determinista.
- **Nodos Satélite: ESP32-S3** para sensores inteligentes distribuidos.

7.2. Pasos Críticos para el Despliegue

1. **Validación de ulab:** Prototipar los algoritmos de Lorenz y TDA simplificado en MicroPython usando ulab para verificar que el rendimiento numérico cumple con los requisitos de tiempo real (ej. < 10ms por ciclo de actualización).
2. **Desarrollo del Driver TDA:** Escribir una extensión C para MicroPython que implemente la filtración Vietoris-Rips optimizada (Ripser-lite) para aprovechar la potencia nativa del P4.
3. **Adopción de Data Mesh:** Definir formalmente los esquemas de datos (Data Contracts) y las APIs de los "Productos de Datos" antes de escribir el código de aplicación, asegurando la interoperabilidad y gobernanza desde el día uno.

Al seguir esta hoja de ruta, APU_filter no solo será un sistema de control funcional, sino una plataforma de vanguardia que demuestra cómo la computación avanzada y los paradigmas de datos modernos pueden converger exitosamente en el borde.

Obras citadas

1. What is a Data Mesh? - Data Mesh Architecture Explained - AWS, fecha de acceso: enero 20, 2026, <https://aws.amazon.com/what-is/data-mesh/>
2. What Is a Data Mesh? - IBM, fecha de acceso: enero 20, 2026, <https://www.ibm.com/think/topics/data-mesh>
3. ESP32-P4 High-performance SoC - Espressif Systems, fecha de acceso: enero 20, 2026, <https://www.espressif.com/en/products/socs/esp32-p4>
4. ESP32-P4 Series - Elecrow, fecha de acceso: enero 20, 2026, https://www.elecrow.com/download/product/DHE04107D/esp32-p4_datasheet_en.pdf
5. ESP32-P4-Pico - Waveshare Wiki, fecha de acceso: enero 20, 2026, <https://www.waveshare.com/wiki/ESP32-P4-Pico>
6. Implementing Sparse Matrices for Graph Algorithms - People @EECS - University of California, Berkeley, fecha de acceso: enero 20, 2026, <https://people.eecs.berkeley.edu/~aydin/GALLA-sparse.pdf>
7. ESP32-P4-WIFI6 - Waveshare Wiki, fecha de acceso: enero 20, 2026, <https://www.waveshare.com/wiki/ESP32-P4-WIFI6>
8. espressif/esp-hosted-mcu - GitHub, fecha de acceso: enero 20, 2026, <https://github.com/espressif/esp-hosted-mcu>
9. ESP32 Hosted - ESPHome - Smart Home Made Simple, fecha de acceso: enero 20, 2026, https://esphome.io/components/esp32_hosted/
10. MIPI DSI Interfaced LCD - ESP32-P4 — ESP-IDF Programming Guide v5.5.2 documentation - Espressif Systems, fecha de acceso: enero 20, 2026, https://docs.espressif.com/projects/esp-idf/en/stable/esp32p4/api-reference/peripherals/lcd/dsi_lcd.html
11. ESP32-S3 vs ESP32-P4 on Matouch 7" display – some practical notes : r/IOT - Reddit, fecha de acceso: enero 20, 2026, https://www.reddit.com/r/IOT/comments/1pv3nct/esp32s3_vs_esp32p4_on_matouch_7_display_some/
12. Technical Comparison Report: ESP32 vs. Teensy 4 Series - Copperhill Technologies, fecha de acceso: enero 20, 2026,

- <https://copperhilltech.com/blog/technical-comparison-report-esp32-vs-teensy-4-series/>
13. ESP32 S3 AI Acceleration Support · micropython · Discussion #14117 - GitHub, fecha de acceso: enero 20, 2026,
<https://github.com/orgs/micropython/discussions/14117>
14. RP2350-Plus 16MB OR ESP32-S3 N16R8 : r/microcontrollers - Reddit, fecha de acceso: enero 20, 2026,
https://www.reddit.com/r/microcontrollers/comments/1jces48/rp2350plus_16mb_or_esp32s3_n16r8/
15. MicroPython Benchmarks - scruss.com, fecha de acceso: enero 20, 2026,
<https://scruss.com/blog/2025/01/21/micropython-benchmarks/>
16. RP2350 and Pico 2 released! - Raspberry Pi Forums, fecha de acceso: enero 20, 2026, <https://forums.raspberrypi.com/viewtopic.php?t=374826>
17. RP2350 - Wikipedia, fecha de acceso: enero 20, 2026,
<https://en.wikipedia.org/wiki/RP2350>
18. A Technical Comparison of the RP2350 and RP2040 Chips - News - SparkFun Electronics, fecha de acceso: enero 20, 2026, <https://news.sparkfun.com/11692>
19. Raspberry Pi Pico 2 W | SC1633 - PiShop.us, fecha de acceso: enero 20, 2026, <https://www.pishop.us/product/raspberry-pi-pico-2-w/>
20. Raspberry Pi Pico 2 W still has no release firmware, 2.5 months after release - Reddit, fecha de acceso: enero 20, 2026,
https://www.reddit.com/r/raspberry_pi/comments/1inb8wf/raspberry_pi_pico_2_w_still_has_no_release/
21. Teensy 4.1 and DSP performance, and possible alternatives, fecha de acceso: enero 20, 2026,
<https://forum.pjrc.com/index.php?threads/teensy-4-1-and-dsp-performance-and-possible-alternatives.66446/>
22. The Lorenz attractor - Scientific Programming with Python, fecha de acceso: enero 20, 2026, <https://scipython.com/blog/the-lorenz-attractor/>
23. Lorenz system - Wikipedia, fecha de acceso: enero 20, 2026,
https://en.wikipedia.org/wiki/Lorenz_system
24. 58 The Control of The Lorenz Chaotic System Blended With The Noise 1. Introduction The initial condition of the state variables, fecha de acceso: enero 20, 2026,
<https://webis.akdeniz.edu.tr/file/getfile?guid=df4a45a5-b5fd-4fc7-a810-193d921c9533>
25. Finding the PID Controller Parameters in Synchronization Between Hyperchaotic Oscillators in QAM (Quadrature Amplitude Modul - ALL SCIENCES PROCEEDINGS, fecha de acceso: enero 20, 2026,
<https://as-proceeding.com/index.php/ijanser/article/download/2887/2742/5582>
26. Designing Pid Controller For Dc Motor By Means Of Chaos - yic.edu.et, fecha de acceso: enero 20, 2026,
https://yic.edu.et/default.aspx/papersCollection/Op1UsL/Designing_Pid_Controller_For_Dc_Motor_By_Means_Of_Chaos.pdf
27. Synchronisation of two chaotic systems using PID control - ResearchGate, fecha

- de acceso: enero 20, 2026,
https://www.researchgate.net/publication/271476933_Synchronisation_of_two_chaotic_systems_using_PID_control
28. Lorenz attractor — Matplotlib 3.10.8 documentation, fecha de acceso: enero 20, 2026, https://matplotlib.org/stable/gallery/mplot3d/lorenz_attractor.html
29. Python, Complex Systems, Chaos and Lorenz Attractor | by Luca Pizziniaco - Medium, fecha de acceso: enero 20, 2026,
<https://medium.com/@lucpiz/python-complex-systems-chaos-and-lorenz-attractor-28499de3f36a>
30. ulab – Manipulate numeric data similar to numpy — Adafruit CircuitPython 1 documentation, fecha de acceso: enero 20, 2026,
<https://docs.circuitpython.org/en/10.0.x/shared-bindings/ulab/index.html>
31. v923z/micropython-ulab: a numpy-like fast vector module for micropython, circuitpython, and their derivatives - GitHub, fecha de acceso: enero 20, 2026,
<https://github.com/v923z/micropython-ulab>
32. Numpy functions — The ulab book 6.11.0 documentation - Read the Docs, fecha de acceso: enero 20, 2026,
<https://micropython-ulab.readthedocs.io/en/latest/numpy-functions.html>
33. An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists - Frontiers, fecha de acceso: enero 20, 2026,
<https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2021.667963/full>
34. Topological data analysis - Wikipedia, fecha de acceso: enero 20, 2026,
https://en.wikipedia.org/wiki/Topological_data_analysis
35. Rips Filtration Tutorial for the R-TDA Package, fecha de acceso: enero 20, 2026,
https://comptag.github.io/rpackage_tutorials//2019/07/tda-rips-tutorial.html
36. vietoris_rips_filtration - Multiscale Spatial Analysis - MuSpAn, fecha de acceso: enero 20, 2026,
https://docs.muspan.co.uk/latest/generated/muspan.topology.vietoris_rips_filtration.html
37. tda-tutorials/Tuto-GUDHI-simplicial-complexes-from-data-points.ipynb at master - GitHub, fecha de acceso: enero 20, 2026,
<https://github.com/MathieuCarriere/tda-tutorials/blob/master/Tuto-GUDHI-simplicial-complexes-from-data-points.ipynb>
38. Streaming algorithm - Wikipedia, fecha de acceso: enero 20, 2026,
https://en.wikipedia.org/wiki/Streaming_algorithm
39. Algorithms for Handling Streaming Data: Mastering Real-Time Data Processing - AlgoCademy Blog, fecha de acceso: enero 20, 2026,
<https://algocademy.com/blog/algorithms-for-handling-streaming-data-mastering-real-time-data-processing/>
40. Phat – Persistent Homology Algorithms Toolbox - Institute of Geometry - TU Graz, fecha de acceso: enero 20, 2026,
https://www.geometrie.tugraz.at/kerber/kerber_papers/phat_jsc.pdf
41. Calculating persistent homology of a simplicial complex - GitHub, fecha de acceso: enero 20, 2026,

https://github.com/Pseudomanifold/Aleph/blob/master/docs/tutorial_persistent_homology_simplicial_complex.md

42. giotto-ai/giotto-ph: High performance implementation of Vietoris-Rips persistence. - GitHub, fecha de acceso: enero 20, 2026, <https://github.com/giotto-ai/giotto-ph>
43. TinyML-based OOD anomaly detection and time synchronization optimization: enhancing the spatiotemporal consistency of anomalous data in industrial IoT - Emerald Publishing, fecha de acceso: enero 20, 2026, <https://www.emerald.com/sr/article/doi/10.1108/SR-07-2025-0503/1320592/TinyML-based-OOD-anomaly-detection-and-time>
44. Multi-Scale Dynamic Graph Learning for Time Series Anomaly Detection (Student Abstract), fecha de acceso: enero 20, 2026, <https://ojs.aaai.org/index.php/AAAI/article/view/30456/32556>
45. Topological Analysis for Detecting Anomalies (TADA) in dependent sequences: application to Time Series. - Journal of Machine Learning Research, fecha de acceso: enero 20, 2026, <http://jmlr.org/papers/volume25/24-0853/24-0853.pdf>
46. TinyML-based OOD anomaly detection and time synchronization optimization: enhancing the spatiotemporal consistency of anomalous data in industrial IoT | Request PDF - ResearchGate, fecha de acceso: enero 20, 2026, https://www.researchgate.net/publication/398287590_TinyML-based_OOD_anomaly_detection_and_time_synchronization_optimization_enhancing_the_spatiotemporal_consistency_of_anomalous_data_in_industrial_IoT
47. How to improve efficiency of my python function involving sparse matrices? - Stack Overflow, fecha de acceso: enero 20, 2026, <https://stackoverflow.com/questions/79424859/how-to-improve-efficiency-of-my-python-function-involving-sparse-matrices>
48. SciPy CSGraph – Compressed Sparse Graph - GeeksforGeeks, fecha de acceso: enero 20, 2026, <https://www.geeksforgeeks.org/python/scipy-csgraph-compressed-sparse-graph/>
49. Easy wireless mesh network with ESP32s and Arduino | Arik Yavilevich's blog, fecha de acceso: enero 20, 2026, <https://blog.yavilevich.com/2025/11/easy-wireless-mesh-network-with-esp32s-and-arduino/>
50. ESP-NOW: Eight Points You Must Know - OpenELAB, fecha de acceso: enero 20, 2026, <https://openelab.io/blogs/learn/esp-now-eight-points-you-must-know>
51. ESP-NOW - ESP32 – ESP-IDF Programming Guide v5.5.2 ..., fecha de acceso: enero 20, 2026, https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html
52. What Is A Data Mesh — And How Not To Mesh It Up - Monte Carlo Data, fecha de acceso: enero 20, 2026, <https://www.montecarlodata.com/blog/what-is-a-data-mesh-and-how-not-to-mesh-it-up/>
53. Demystifying data mesh - Data.world, fecha de acceso: enero 20, 2026,

<https://data.world/blog/demystifying-data-mesh/>

54. Data Mesh Principles: Concepts, Implementation, and Best Practices - Acceldata, fecha de acceso: enero 20, 2026,
<https://www.acceldata.io/blog/top-data-mesh-principles-to-transform-your-data-management>
55. The 4 principles of data mesh | dbt Labs, fecha de acceso: enero 20, 2026,
<https://www.getdbt.com/blog/the-four-principles-of-data-mesh>
56. What is Data Mesh Architecture? Principles & Components - Atlan, fecha de acceso: enero 20, 2026,
<https://atlan.com/understanding-data-mesh-architecture/>
57. Observability in the context of Data Mesh — Part 1 | by P Platter | Agile Lab Engineering, fecha de acceso: enero 20, 2026,
<https://p-platter.medium.com/observability-in-the-context-of-data-mesh-part-1-792e2b328489>
58. Memory use in CPython and MicroPython - LWN.net, fecha de acceso: enero 20, 2026, <https://lwn.net/Articles/725508/>
59. CircuitPython/CircuitPython2026.md at main - GitHub, fecha de acceso: enero 20, 2026,
<https://github.com/TheKitty/CircuitPython/blob/main/CircuitPython2026.md>
60. Python on Microcontrollers Newsletter: Original Pi Heading Towards EOL, CircuitPython MIPI DSI, LLM Robots and More! #CircuitPython #Python #micropython @ThePSF @Raspberry_Pi - Adafruit Daily, fecha de acceso: enero 20, 2026,
https://www.adafruitdaily.com/2025/11/10/python-on-microcontrollers-newsletter-original-pi-heading-towards-eol-circuitpython-mipi-dsi-lm-robots-and-more-circuitpython-micropython-thedspf-raspberry_pi/
61. mipidsi – Low-level routines for interacting with MIPI DSI - CircuitPython, fecha de acceso: enero 20, 2026,
<https://docs.circuitpython.org/en/latest/shared-bindings/mipidsi/index.html>
62. NetworkX — NetworkX documentation, fecha de acceso: enero 20, 2026,
<https://networkx.org/>