

Informe de Viabilidad Técnica: Arquitectura Cognitiva Híbrida Fractal para Robótica en el Borde

1. Introducción y Contexto de la Investigación

1.1 El Imperativo del Cambio de Paradigma en la IA Corporizada

La Inteligencia Artificial Corporizada (Embodied AI) se encuentra en una encrucijada crítica. Los paradigmas dominantes, basados fundamentalmente en el Aprendizaje Profundo (Deep Learning) clásico y modelos masivos como los Transformers o grandes CNNs (Redes Neuronales Convolucionales), han demostrado una capacidad excepcional para el reconocimiento de patrones estáticos y el procesamiento de lenguaje natural. Sin embargo, su aplicación directa en robótica autónoma, particularmente en hardware de borde con restricciones energéticas y computacionales severas como el **NVIDIA Jetson Nano**, presenta barreras técnicas insalvables.¹

El enfoque clásico de "End-to-End" mediante retropropagación del error (Backpropagation) impone una latencia inaceptable para los bucles sensoriomotores en tiempo real y exige un consumo energético que drena rápidamente las baterías de plataformas móviles. Más criticamente, estos modelos carecen a menudo de la plasticidad necesaria para adaptarse *online* a entornos dinámicos sin sufrir el olvido catastrófico.³

Este informe propone y valida técnicamente una alternativa radical: una **Arquitectura Cognitiva Híbrida**. Esta arquitectura se aleja de la fuerza bruta computacional para abrazar principios de la **Neurociencia Computacional** y la **Cibernética de Segundo Orden**. En lugar de entrenar redes densas mediante descenso de gradiente estocástico (SGD) masivo, proponemos un sistema que aprovecha la dinámica de sistemas complejos (Reservoir Computing), la eficiencia de la codificación fractal (Curvas de Hilbert) y mecanismos de aprendizaje inspirados en el desarrollo biológico (Motor Babbling, Homeostasis).

1.2 Definición del Problema: Restricciones del Hardware de Borde

El dispositivo objetivo, el NVIDIA Jetson Nano, define el espacio de diseño. Con una GPU de arquitectura Maxwell (128 CUDA cores) y 4GB de RAM compartida, el Nano no es apto para el *entrenamiento* de modelos profundos modernos, aunque es competente en inferencia.² Sin embargo, su capacidad para realizar operaciones matriciales paralelizadas mediante CUDA lo convierte en un sustrato ideal para arquitecturas que dependen de multiplicaciones de matrices dispersas, como las Echo State Networks (ESN).⁵

La investigación sugiere que la limitación no es la capacidad de cómputo *per se*, sino cómo se

utiliza. Mientras que una CNN desperdicia ciclos en convoluciones redundantes sobre vóxeles vacíos o píxeles estáticos⁷, una arquitectura basada en **Curvas de Relleno del Espacio (Space-Filling Curves, SFC)** y **Computación de Reservorio** puede procesar flujos de información temporal con una fracción del costo energético, manteniendo una representación rica del estado del mundo.⁸

1.3 Visión General de la Arquitectura Propuesta

El sistema se estructura en tres capas jerárquicas pero acopladas dinámicamente, diseñadas para emular la organización del sistema nervioso central:

1. **Capa Periférica (Fusión Sensorial Fractal):** Transducción eficiente de datos espaciales (visión, LiDAR) a secuencias temporales mediante SFCs (Hilbert), preservando la localidad sin el costo de las convoluciones 3D.⁸
2. **Núcleo Cognitivo (Dinámica Emergente):** Un sustrato de memoria dinámica basado en Reservoir Computing (RC), específicamente Deep ESNs con topología fractal, que opera en el "borde del caos" para integrar información multisensorial y generar dinámicas atractores.¹¹
3. **Capa de Aprendizaje (Desarrollo y Homeostasis):** Un sistema de motivación intrínseca basado en la regulación homeostática (minimización de entropía interna) que guía la exploración motora (Babbling) y consolida el aprendizaje mediante reglas Hebbianas locales no supervisadas.¹³

2. Capa Periférica: Fusión Sensorial Fractal y Curvas de Relleno del Espacio

La primera barrera para la IA en el borde es la "maldición de la dimensionalidad". Los sensores modernos, como cámaras de profundidad o LiDAR, generan nubes de puntos o matrices de píxeles masivas. Procesar esto con CNNs 3D es computacionalmente prohibitivo para el Jetson Nano.⁷ La solución propuesta reside en la **linealización fractal**.

2.1 Teoría de las Curvas de Relleno del Espacio (SFC)

Las Curvas de Relleno del Espacio, descubiertas por Peano y Hilbert, son funciones continuas y sobreyectivas que mapean un intervalo unitario unidimensional (tiempo/secuencia) a un hipercubo multidimensional (espacio/imagen). Matemáticamente, una curva de Hilbert

$H : \rightarrow^n$ posee propiedades críticas para la robótica:

1. **Preservación de la Localidad:** Puntos que están cercanos en el espacio 2D/3D tienden a estar cercanos en la secuencia 1D. Esto es fundamental para que el Núcleo Cognitivo (que es una red recurrente temporal) pueda inferir relaciones espaciales a partir de la secuencia temporal.⁷

2. **Autosemejanza Fractal:** La curva se construye recursivamente. Esto permite un procesamiento **multirresolución** natural. El sistema puede procesar una escena a baja resolución (orden de curva bajo) para una reacción rápida y refinar zonas de interés con un orden superior, sin cambiar la arquitectura de la red neuronal subyacente.¹⁶

2.1.1 Análisis Comparativo de SFCs para Visión Robótica

Al evaluar las opciones para la linealización de imágenes en el Jetson Nano, la literatura favorece la Curva de Hilbert sobre la Curva-Z (Morton) o el escaneo raster tradicional.

Característica	Escaneo Raster (Lineal)	Curva-Z (Morton)	Curva de Hilbert
Complejidad de Cálculo	Trivial ($O(1)$)	Baja (Bit interleaving)	Media (Bitwise + Rotación)
Preservación de Localidad	Pobre (Saltos de fila rompen vecindad vertical)	Buena, pero con discontinuidades diagonales	Excelente (Sin saltos severos)
Coherencia Temporal	Baja	Media	Alta (Ideal para RNNs/ESNs)
Costo en Hardware	Nulo	Muy bajo (ALU simple)	Bajo (Tabla de búsqueda o lógica simple)

Insight de Segundo Orden: La superioridad de la curva de Hilbert radica en que maximiza la autocorrelación de la señal 1D resultante. Para un sistema de Reservoir Computing, que funciona como un sistema dinámico con memoria de desvanecimiento, una señal de entrada con alta autocorrelación local facilita la predicción y el filtrado de ruido. Al transformar la estructura espacial de una imagen en estructura temporal, permitimos que el reservorio "vea" el espacio a través del tiempo, una estrategia biomimética similar a las sacadas oculares.¹⁷

2.2 Algoritmos de Mapeo y Fusión Multisensorial

Para la implementación en el Jetson Nano, no se debe calcular la geometría de la curva en cada frame. En su lugar, se utilizan **Tablas de Búsqueda (Lookup Tables - LUT)**

precalculadas que mapean coordenadas (x, y) a índices d de Hilbert.

El proceso de **Fusión Sensorial Fractal** propuesto integra visión (RGB) y profundidad (D) o

Lidar en una sola trama entrelazada:

1. **Cuantización:** La imagen se reduce a una resolución base de $N \times N$ (donde $N = 2^k$).
2. **Mapeo:** Cada píxel (i, j) se asigna a su índice $h = Hilbert(i, j)$.
3. **Serialización:** Los valores de los sensores (R, G, B, Profundidad) se ordenan según h .
4. **Inyección:** El vector resultante $u(t)$ alimenta al reservorio.

Este método evita la necesidad de redes separadas para procesar distintas modalidades y luego fusionarlas (Late Fusion), reduciendo drásticamente la carga de memoria.¹⁹ La investigación²¹ demuestra que usar SFCs para serializar imágenes permite a modelos secuenciales (como Mamba o RNNs) lograr un rendimiento competitivo con un costo computacional lineal, en contraste con el costo cuadrático de los mecanismos de atención global en Vision Transformers.

2.3 Implementación Eficiente en Hardware de Borde

Aunque el cálculo del índice de Hilbert es recursivo conceptualmente, en la práctica se implementa mediante operaciones a nivel de bit (bitwise operators) que son extremadamente rápidas en la CPU ARM Cortex-A57 del Jetson Nano, o incluso paralelizables en la GPU mediante kernels CUDA simples.²²

Pseudocódigo Optimizado (Python/Numba para Jetson):

Python

```
import numpy as np
from numba import jit

@jit(nopython=True)
def hilbert_index(x, y, order):
    d = 0
    s = 2**order - 1
    while s > 0:
        rx = (x & s) > 0
        ry = (y & s) > 0
        d += s * s * ((3 * rx) ^ ry)
        if ry == 0:
```

```

if rx == 1:
    x = (2**order - 1) - x
    y = (2**order - 1) - y
    # Swap x and y
    x, y = y, x
    s //= 2
return d

```

Este código se ejecuta pre-calculado o compilado JIT para minimizar latencia.

La literatura¹⁵ sugiere que este enfoque de "Deep Space Filling Curves" puede reducir el uso de memoria de la GPU significativamente, permitiendo procesar nubes de puntos o véxeles que de otro modo desbordarían los 4GB del Nano.

3. Núcleo Cognitivo: Reservoir Computing en el Borde

El corazón de la arquitectura propuesta es el **Reservoir Computing (RC)**, específicamente las **Echo State Networks (ESN)**. A diferencia del Deep Learning, donde se entrena todos los pesos, en una ESN solo se entrena la capa de salida (Readout). El "reservorio" es una red recurrente grande, fija y dispersa que actúa como un medio excitante no lineal.³

3.1 Justificación Técnica para el Jetson Nano

El RC es la solución óptima para el hardware de borde por tres razones fundamentales derivadas de los *snippets* de investigación:

1. **Entrenamiento Lineal:** El entrenamiento se reduce a una regresión lineal (Ridge Regression), que tiene solución analítica cerrada. Esto elimina la necesidad de retropropagación a través del tiempo (BPTT), reduciendo la carga computacional en órdenes de magnitud.³
2. **Operaciones Dispersas:** La matriz de pesos del reservorio W suele tener una dispersión (sparsity) del 95-99%. El Jetson Nano, aunque limitado en FLOPs brutos, puede manejar multiplicaciones de matrices dispersas (SpMV) de manera eficiente utilizando librerías como cuSPARSE o a través de CuPy.⁶
3. **Memoria de Corto Plazo (Echo State Property):** El reservorio integra información temporal de manera inherente. Esto es crucial para la robótica, donde el estado actual depende de la trayectoria histórica (inerzia, momento), algo que las redes feedforward clásicas no capturan sin arquitecturas complejas de ventanas deslizantes.²⁵

3.2 Arquitectura Fractal del Reservorio

Para potenciar la capacidad del reservorio más allá de una red aleatoria estándar

(Erdős–Rényi), proponemos una **topología fractal** o jerárquica. La investigación ¹¹ indica que las redes neuronales con conectividad fractal o de "pequeño mundo" (Small-World) exhiben dinámicas en múltiples escalas de tiempo.

- **Multiescala Temporal:** Un robot necesita reaccionar a perturbaciones rápidas (ms) mientras mantiene un objetivo a largo plazo (segundos/minutos). Un reservorio estándar tiene una constante de tiempo uniforme. Un **Reservoir Fractal** estructura sus conexiones para crear clústeres de neuronas con dinámicas rápidas y clústeres con dinámicas lentas, desacopladas pero comunicadas.²⁸
- **Dimensión Fractal y Capacidad de Memoria:** Existe una correlación teórica entre la dimensión fractal de la señal de entrada (la curva de Hilbert) y la conectividad óptima del reservorio. Al alinear la dimensión fractal de la topología de red con la de los datos de entrada, se maximiza la capacidad de memoria y la separabilidad lineal del reservorio.²⁷

Ecuación de Actualización de Estado (Leaky-Integrator ESN):

La dinámica de las neuronas en el Jetson Nano se rige por la ecuación de integrador con fugas (Leaky-Integrator), que permite ajustar la "velocidad" de la memoria:

$$\mathbf{x}(t+1) = (1 - \alpha)\mathbf{x}(t) + \alpha \tanh(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t) + \mathbf{W}_{fb}\mathbf{y}(t))$$

Donde:

- $\mathbf{x}(t)$: Estado interno del reservorio (vector de alta dimensión, e.g., 1000-5000 neuronas).
- $\mathbf{u}(t)$: Entrada sensorial (vector de la Curva de Hilbert).
- \mathbf{W} : Matriz de pesos recurrentes (Fija, dispersa, espectralmente calibrada).
- α : Tasa de fuga (Leaking rate), controla la inercia del estado.

3.3 Stack Tecnológico: Python, CuPy y ReservoirPy

La implementación recomendada utiliza **Python** como lenguaje de orquestación, apoyado por librerías que descargan el cómputo pesado a la GPU y a C++.

1. **ReservoirPy:** Es la librería de referencia seleccionada. Permite la creación flexible de ESNs, entrenamiento online y offline, y lo más importante, tiene soporte experimental y creciente para backends acelerados.³¹
2. **CuPy:** Para el Jetson Nano, la aceleración por GPU es obligatoria para reservorios grandes (>1000 neuronas). **CuPy** ofrece una interfaz tipo NumPy pero ejecuta las operaciones en la GPU mediante CUDA. Las operaciones clave como `cp.sparse.csr_matrix.dot` (multiplicación dispersa) son vitales para mantener la viabilidad en tiempo real.³³

3. **Integración ROS2:** El sistema debe encapsularse en nodos ROS2 (Robot Operating System 2), preferiblemente en su versión Foxy o Humble (compatibles con Ubuntu 20.04/18.04). ROS2 permite una gestión eficiente de mensajes entre la periferia (cámara) y el núcleo cognitivo, con menor overhead que ROS1.³⁵
-

4. Capa de Aprendizaje: Desarrollo, Homeostasis y Dinámica

Esta capa distingue al sistema propuesto de un simple controlador reactivo. Aquí es donde surge la autonomía verdadera, inspirada en la robótica del desarrollo (Developmental Robotics) y teorías biológicas.

4.1 Robótica del Desarrollo: Del Caos al Control

En lugar de programar explícitamente la cinemática inversa (IK) del robot, el sistema la aprende. Este enfoque, conocido como adquisición del esquema corporal, permite al robot adaptarse a daños (e.g., un motor atascado) re-aprendiendo su modelo interno.¹³

4.1.1 Motor Babbling (Balbuceo Motor)

El proceso comienza con el **Motor Babbling**. El robot genera comandos motores aleatorios y observa los cambios resultantes en sus sensores (Periferia).

- **Fase 1: Balbuceo Aleatorio:** Exploración estocástica del espacio de acciones. Genera un dataset inicial de pares (Acción, Efecto).³⁷
- **Fase 2: Goal Babbling:** Una vez que se tiene un modelo preliminar, el robot deja de explorar acciones aleatorias y comienza a explorar objetivos sensoriales aleatorios. Intenta alcanzar un estado sensorial deseado usando su modelo inverso incipiente. Esto reduce drásticamente el espacio de búsqueda en sistemas redundantes.³⁶

La investigación ³⁹ muestra que el *Goal Babbling* permite aprender la cinemática inversa de brazos robóticos de alta dimensionalidad con solo unos cientos de movimientos, haciéndolo viable para el aprendizaje *online* en el Jetson Nano.

4.2 Homeostasis Digital y Motivación Intrínseca

¿Qué impulsa al robot a moverse si no hay una tarea externa? La **Homeostasis**. El sistema mantiene un conjunto de Variables Esenciales (VE) dentro de un rango viable. Estas pueden ser físicas (nivel de batería, temperatura de motores) o cognitivas (incertidumbre de predicción).⁴⁰

Teoría de Reducción de Impulso (Drive Reduction):

El "dolor" o "impulso" (D) se define como la desviación de las VEs de sus puntos de ajuste (Setpoints). La "recompensa" intrínseca es la reducción de este impulso.

$$D(t) = \sum_i |v_i(t) - setpoint_i|$$

El objetivo del sistema de aprendizaje es minimizar $\frac{dD}{dt}$.

Curiosidad como Homeostasis de la Información: Para evitar que el robot se quede quieto (lo que minimiza el gasto energético pero no el aprendizaje), introducimos una VE de "Entropía de Predicción". Si el reservorio predice perfectamente el entorno, la "incertidumbre" cae por debajo del setpoint, generando un impulso de "aburrimiento". Esto fuerza al robot a explorar situaciones nuevas (aumentar la incertidumbre momentáneamente) para luego aprenderlas (reducirla), un proceso alineado con el **Principio de Energía Libre** (Free Energy Principle / Active Inference).⁴²

4.3 Aprendizaje Hebbiano y Dinámica de Atractores

Mientras que el *readout* del reservorio se entrena mediante regresión supervisada (generada por el propio babbling), las conexiones internas y periféricas se pueden refinar mediante **Aprendizaje Hebbiano** no supervisado.

Regla de Oja: Para evitar que los pesos crezcan infinitamente (un problema del Hebbiano clásico), se utiliza la Regla de Oja, que incluye un término de normalización. Esto permite que las neuronas del reservorio se especialicen en detectar "primitivas motoras" o patrones sensoriales frecuentes (análisis de componentes principales *online*).⁴⁴

$$\Delta w_{ij} = \eta \cdot y_i \cdot (x_j - y_i \cdot w_{ij})$$

Donde y es la salida de la neurona, x la entrada, y η la tasa de aprendizaje. Esto se puede implementar eficientemente en el Nano como operaciones vectoriales locales.

Dinámica de Atractores: El comportamiento estable (e.g., caminar, mantenerse en pie) no se almacena como una "grabación", sino como un **Atractor** en el espacio de fases del reservorio. El entrenamiento busca moldear el paisaje energético del reservorio para que los estados deseados sean cuencas de atracción estables. Si el robot es empujado (perturbación), la dinámica del sistema lo devuelve naturalmente al atractor (ciclo límite), proporcionando robustez física sin necesidad de cómputo de control de alta frecuencia.¹²

5. Especificación del Stack Tecnológico Recomendado

Para materializar esta arquitectura en un NVIDIA Jetson Nano, se recomienda la siguiente pila tecnológica, seleccionada por su eficiencia y compatibilidad con la investigación analizada.

Capa	Tecnología / Librería	Justificación basada en Investigación
OS	Ubuntu 18.04 LTS (JetPack 4.6)	Soporte oficial más estable para drivers CUDA en Nano. ⁴⁷
Middleware	ROS 2 Foxy Fitzroy	Comunicación eficiente, soporte Python 3 nativo, mejor tiempo real que ROS1. ³⁵
Backend Numérico	CuPy	Aceleración GPU de operaciones NumPy. Crítico para SpMV en ESNs. ³³
Reservoir Computing	ReservoirPy	Diseñada específicamente para RC, flexible, soporte para entrenamiento online y Scikit-learn API. ³¹
Visión	OpenCV (con CUDA)	Preprocesamiento de imágenes rápido antes del mapeo Hilbert. Debe compilarse con soporte CUDA. ⁴⁹
Curvas SFC	Implementación Custom (Numba)	Librerías estándar son lentas. Se requiere implementación JIT o C++ binded. ²²
Simulación	Genesis / Gazebo	Genesis (Embodied AI) para pre-entrenamiento físico rápido; Gazebo para integración ROS clásica. ⁵⁰
Aprendizaje Online	FORCE / RLS	Algoritmos de aprendizaje

		recursivo para adaptar el <i>readout</i> en tiempo real. ³
--	--	---

5.1 Consideraciones de Instalación en Jetson Nano

Instalar este stack en el Nano no es trivial debido a la arquitectura ARM64 (aarch64).

- **CuPy:** No se puede instalar simplemente con pip install cupy. Se requiere compilar desde la fuente o buscar wheels precompilados específicos para JetPack, ya que depende de versiones específicas de CUDA Toolkit (v10.2 en JetPack 4.6).³³
- **OpenCV:** La versión preinstalada en JetPack a menudo no tiene soporte CUDA para Python habilitado. Se recomienda usar scripts de compilación automática (e.g., de JetsonHacks) para habilitar las optimizaciones de hardware.⁵³
- **Memoria:** Con solo 4GB, es **crítico** configurar una partición SWAP de al menos 4-6GB (preferiblemente en un SSD NVMe USB 3.0) para evitar que el compilador o el runtime maten los procesos por *Out Of Memory* (OOM).⁵⁴

6. Algoritmos y Pseudocódigo Detallado

A continuación, se presenta la lógica algorítmica central para la implementación del bucle cognitivo.

6.1 Bucle Principal: Percepción-Acción-Aprendizaje

Este pseudocódigo ilustra cómo se integran la curva de Hilbert, el reservorio y la homeostasis.

Python

```
# Pseudocódigo de Arquitectura Híbrida en Jetson Nano
import cupy as cp
from reservoirpy.nodes import Reservoir, Ridge
from custom_hilbert import hilbert_encode_gpu
from homeostasis import calculate_drive

class CognitiveBot:
    def __init__(self):
        # 1. Configuración del Reservorio (Núcleo Cognitivo)
        # N=1000 neuronas es viable en Nano con CuPy
        self.reservoir = Reservoir(units=1000, lr=0.2, sr=0.95,
                                   rc_connectivity=0.01) # 1% de densidad (Sparse)
```

```

# Readout entrenable (Mapea estado del reservorio a motores)
self.readout = Ridge(output_dim=4, ridge=1e-5)
self.esn = self.reservoir >> self.readout

# 2. Estado Homeostático
self.energy_level = 1.0
self.entropy = 0.0
self.is_babbling = True

def process_frame(self, image_gpu):
    """
    Paso 1: Fusión Sensorial Fractal (Periferia)
    image_gpu: Array CuPy en VRAM
    """

    # Downsampling y codificación Hilbert (Kernel CUDA)
    # Transforma imagen 2D -> Secuencia temporal fractal 1D
    sensory_stream = hilbert_encode_gpu(image_gpu, order=4)

    # Paso 2: Dinámica del Reservorio (Núcleo)
    # Actualización de estado: x(t+1) = tanh(Win*u + W*x)
    # Ejecutado en GPU via CuPy
    reservoir_state = self.reservoir(sensory_stream)

    # Paso 3: Decisión y Aprendizaje (Homeostasis)
    drive = calculate_drive(self.energy_level, self.entropy)

    if self.is_babbling or drive > THRESHOLD:
        # Modo Exploración: Motor Babbling
        # Generar ruido fractal (no ruido blanco) para movimientos naturales
        motor_cmd = self.generate_fractal_noise()

        # Aprendizaje Online (FORCE / RLS)
        # Entrenar el readout para asociar el estado actual con el comando ejecutado
        # (Modelo Inverso: Estado -> Acción)
        self.readout.fit(reservoir_state, motor_cmd)

        # Actualización Hebbiana de pesos internos (Opcional, más lento)
        # self.apply_hebbian_plasticity(self.reservoir)

    else:
        # Modo Explotación: Usar lo aprendido
        motor_cmd = self.esn(sensory_stream)

```

```

    return motor_cmd

def generate_fractal_noise(self):
    # Ruido 1/f (Pink Noise) para exploración motora biomimética
    # Evita sacudidas bruscas dañinas para los servos
    pass

```

6.2 Implementación de Aprendizaje Online (FORCE Learning Simplificado)

Para que el Nano aprenda en tiempo real, no podemos acumular grandes lotes de datos. Usamos algoritmos recursivos. El algoritmo **FORCE** (First-Order Reduced and Controlled Error) o RLS es ideal.³

Python

```

def train_online_rls(P, w_out, state, target, alpha=1.0):
    """
    Actualización de pesos de salida usando Recursive Least Squares (RLS).
    P: Matriz de covarianza inversa (aprox)
    w_out: Pesos de salida actuales
    state: Estado del reservorio x(t)
    target: Señal objetivo (e.g., comando motor real ejecutado durante babbling)
    """

    # Cálculo del error de predicción
    prediction = cp.dot(w_out, state)
    error = prediction - target

    # Actualización de la matriz P (Ganancia)
    k = cp.dot(P, state) / (alpha + cp.dot(state.T, cp.dot(P, state)))
    P = (P - cp.dot(k, cp.dot(state.T, P))) / alpha

    # Actualización de pesos
    w_out = w_out - cp.dot(error, k.T)

return w_out, P

```

Nota: En el Jetson Nano, la actualización de la matriz P ($N \times N$) es costosa ($O(N^2)$). Se recomienda actualizar los pesos solo periódicamente o cuando el error de predicción

(sorpresa) supere un umbral homeostático, ahorrando ciclos de GPU.

7. Análisis de Papers y Referencias Académicas Clave

La investigación se sustenta en una selección rigurosa de literatura académica. A continuación, se detallan los trabajos más influyentes para esta arquitectura y su relevancia específica.

7.1 Fusión Sensorial y SFCs

- ⁸ Hilbert curve for dimensionality reduction: Propone el uso de curvas de Hilbert para destilar conocimiento de redes 3D a 2D. Relevancia: Valida teóricamente el uso de SFC para preservar información estructural en representaciones de menor dimensión.
- ⁷ HilbertNet: Demuestra que aplatar nubes de puntos 3D con curvas de Hilbert preserva la localidad mejor que la voxelización, reduciendo el costo computacional. Fundamental para la capa periférica.
- ¹⁵ Efficient Deep Space Filling Curve: Muestra cómo las SFC pueden optimizarse mediante aprendizaje profundo, superando a las curvas estáticas en tareas de visión.

7.2 Reservoir Computing y Hardware

- ³ Practical Echo State Networks (Jaeger): La "biblia" práctica de las ESNs. Proporciona los trucos de ingeniería necesarios para que el reservorio no colapse ni sature (escalado de entradas, radio espectral).
- ⁹ Reservoir Computing edge hardware efficiency: Analiza el trade-off entre tamaño del reservorio y consumo energético en FPGAs y hardware de borde. Confirma la viabilidad de RC frente a CNNs.
- ³⁰ Fractal dimension of reservoir computers: Estudio crucial que vincula la dimensión fractal de la señal de entrada con el rendimiento del reservorio. Sugiere que el reservorio debe operar en una dimensión fraccionaria efectiva para maximizar el procesamiento de información.

7.3 Robótica del Desarrollo y Homeostasis

- ³⁸ Goal Babbling permits direct learning of inverse kinematics: Paper seminal que demuestra la eficiencia del Goal Babbling sobre el Motor Babbling puro para espacios de alta dimensión.
- ¹⁴ Continuous Homeostatic Reinforcement Learning: Introduce el marco matemático HRRL (Homeostatic Regulated RL), uniendo la teoría de control y el aprendizaje por refuerzo. Proporciona las ecuaciones para la función de impulso homeostático.
- ⁵⁵ Behavior-driven synaptic plasticity (Hebbian): Propone una regla sináptica ("chaining together what changes together") que permite la emergencia de competencias

sensoriomotoras sin supervisión externa.

8. Conclusiones y Viabilidad Técnica

El análisis exhaustivo de los algoritmos y el hardware confirma que la construcción de una **Arquitectura Cognitiva Híbrida** en el NVIDIA Jetson Nano es **técnicamente viable** y ofrece ventajas estratégicas sobre el Deep Learning convencional para la robótica autónoma.

1. **Eficiencia Computacional:** Al sustituir las convoluciones densas por codificación fractal (SFC) y el backpropagation por regresión lineal (RC/RLS), se estima una reducción de órdenes de magnitud en los FLOPs requeridos para el aprendizaje *online*.
2. **Adaptabilidad:** La inclusión de mecanismos homeostáticos y de *babbling* permite al robot adaptarse a cambios en su cuerpo (daños) o entorno sin necesidad de reentrenamiento externo masivo, abordando el problema del "Sim-to-Real".
3. **Limitaciones:** La memoria RAM (4GB) es el cuello de botella principal. El uso de matrices dispersas (CuPy sparse matrices) y la gestión agresiva de memoria (SWAP) son obligatorios. Además, la calidad de la visión dependerá fuertemente de la resolución de la curva de Hilbert; resoluciones muy bajas podrían perder detalles críticos para la navegación fina.

Recomendación Final: Se recomienda proceder con un prototipo que implemente primero la capa de Reservoir Computing con *Motor Babbling* simple en un entorno simulado (Genesis/Gazebo) desplegado en el Nano, antes de integrar la visión completa mediante curvas de Hilbert, para aislar y validar la dinámica de aprendizaje homeostático.

Works cited

1. AI on the Road: NVIDIA Jetson Nano-Powered Computer Vision-Based System for Real-Time Pedestrian and Priority Sign Detection - MDPI, accessed February 2, 2026, <https://www.mdpi.com/2076-3417/14/4/1440>
2. Can I use Jetson Nano to train a neural network? - NVIDIA Developer Forums, accessed February 2, 2026, <https://forums.developer.nvidia.com/t/can-i-use-jetson-nano-to-train-a-neural-network/73521>
3. A practical guide to applying echo state networks - Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, accessed February 2, 2026, <https://www.ai.rug.nl/minds/uploads/PracticalESN.pdf>
4. Efficient Learning in Neural Networks without Gradient Backpropagation - OpenReview, accessed February 2, 2026, <https://openreview.net/forum?id=ROYGjmqiwB>
5. High-performance Echo State Network simulation, optimization and visualization in modern C++. - GitHub, accessed February 2, 2026, <https://github.com/FloopCZ/echo-state-networks>
6. Efficient Sparse Matrix-Vector Multiplication on CUDA - NVIDIA, accessed

- February 2, 2026, <https://www.nvidia.com/docs/io/66889/nvr-2008-004.pdf>
- 7. Efficient Point Cloud Analysis Using Hilbert Curve, accessed February 2, 2026, https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136620717.pdf
 - 8. Hilbert Distillation for Cross-Dimensionality Networks - NeurIPS, accessed February 2, 2026, https://proceedings.neurips.cc/paper_files/paper/2022/file/4c9477b9e2c7ec0ad3f4f15077aaaf85a-Paper-Conference.pdf
 - 9. Hardware-Optimized Reservoir Computing System for Edge Intelligence Applications - UIB, accessed February 2, 2026, <https://repositori.uib.es/xmlui/bitstream/handle/11201/164122/559150.pdf?sequence=1&isAllowed=y>
 - 10. Analysis of Multi-Dimensional Space-Filling Curves, accessed February 2, 2026, <https://people.cs.vt.edu/ctlu/Course/2025F/CS6604/SFile/SFC-Analysis-HW2.pdf>
 - 11. [2105.06923] Hierarchical Architectures in Reservoir Computing Systems - arXiv, accessed February 2, 2026, <https://arxiv.org/abs/2105.06923>
 - 12. Motor control in a meta-network with attractor dynamics - PubMed, accessed February 2, 2026, <https://pubmed.ncbi.nlm.nih.gov/17925260/>
 - 13. Goal-related feedback guides motor exploration and redundancy resolution in human motor skill acquisition - Research journals - PLOS, accessed February 2, 2026, <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006676>
 - 14. (PDF) Continuous Homeostatic Reinforcement Learning for Self-Regulated Autonomous Agents - ResearchGate, accessed February 2, 2026, https://www.researchgate.net/publication/354597646_Continuous_Homeostatic_Reinforcement_Learning_for_Self-Regulated_Autonomous_Agents
 - 15. Efficient Deep Space Filling Curve - CVF Open Access, accessed February 2, 2026, https://openaccess.thecvf.com/content/ICCV2023/papers/Chen_Efficient_Deep_Space_Filling_Curve_ICCV_2023_paper.pdf
 - 16. Constructing Hierarchical Continuity in Hilbert & Moore Treemaps - Eurographics, accessed February 2, 2026, <https://diglib.eg.org/bitstreams/be1ebd9e-423e-4178-8531-1a18930229b7/download>
 - 17. An Adaptive Space-Filling Curve Trajectory for Ordering 3D Datasets to 1D: Application to Brain Magnetic Resonance Imaging Data for Classification - NIH, accessed February 2, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC7304044/>
 - 18. Hilbert Curve Based Molecular Sequence Analysis - arXiv, accessed February 2, 2026, <https://arxiv.org/html/2412.20616v1>
 - 19. Performance of Multi-Dimensional Space- Filling Curves - Purdue e-Pubs, accessed February 2, 2026, <https://docs.lib.psu.edu/cgi/viewcontent.cgi?article=2545&context=cstech>
 - 20. (PDF) Analysis of Multi-Dimensional Space-Filling Curves - ResearchGate, accessed February 2, 2026, https://www.researchgate.net/publication/220387671_Analysis_of_Multi-Dimensional_Space-Filling_Curves

21. {cutout}00.3cm22cm1 empty PointMamba: A Simple State Space Model for Point Cloud Analysis - arXiv, accessed February 2, 2026,
<https://arxiv.org/html/2402.10739v5>
22. Mapping N-dimensional value to a point on Hilbert curve - Stack Overflow, accessed February 2, 2026,
<https://stackoverflow.com/questions/499166/mapping-n-dimensional-value-to-a-point-on-hilbert-curve>
23. Reservoir computing - Wikipedia, accessed February 2, 2026,
https://en.wikipedia.org/wiki/Reservoir_computing
24. Leveraging Memory Copy Overlap for Efficient Sparse Matrix-Vector Multiplication on GPUs, accessed February 2, 2026,
<https://www.mdpi.com/2079-9292/12/17/3687>
25. An Echo State Network Imparts a Curve Fitting - University of Pretoria, accessed February 2, 2026,
<https://repository.up.ac.za/server/api/core/bitstreams/0db39579-3027-4226-bd5c-8e27910623d0/content>
26. Optimization and Applications of Echo State Networks with Leaky Integrator Neurons - Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, accessed February 2, 2026,
<https://www.ai.rug.nl/minds/uploads/leakyESN.pdf>
27. Analyzing Echo-state Networks Using Fractal Dimension - arXiv, accessed February 2, 2026, <https://arxiv.org/pdf/2205.09348>
28. Discovering multiscale dynamical features with hierarchical Echo State Networks1 Technical Report No. 10, accessed February 2, 2026,
https://www.ai.rug.nl/minds/uploads/hierarchicalesn_techrep10.pdf
29. Analyzing Echo-state Networks Using Fractal Dimension - IEEE Xplore, accessed February 2, 2026, <https://ieeexplore.ieee.org/document/9892199/>
30. Dimension of Reservoir Computers | alphaXiv, accessed February 2, 2026,
<https://www.alphaxiv.org/overview/1912.06472v1>
31. ReservoirPy — ReservoirPy 0.4.1 documentation, accessed February 2, 2026,
<https://reservoirpy.readthedocs.io/>
32. A quick start to ReservoirPy — ReservoirPy 0.4.1 documentation, accessed February 2, 2026,
https://reservoirpy.readthedocs.io/en/latest/user_guide/quickstart.html
33. cupy/cupy: NumPy & SciPy for GPU - GitHub, accessed February 2, 2026,
<https://github.com/cupy/cupy>
34. CuPy: NumPy & SciPy for GPU, accessed February 2, 2026, <https://cupy.dev/>
35. Using Python Packages with ROS 2 - ROS documentation, accessed February 2, 2026, <https://docs.ros.org/en/foxy/How-To-Guides/Using-Python-Packages.html>
36. Online Goal Babbling for rapid bootstrapping of inverse models in high dimensions, accessed February 2, 2026,
<https://www.honda-ri.de/pubs/pdf/1732.pdf>
37. A Bio-Inspired Mechanism for Learning Robot Motion From Mirrored Human Demonstrations, accessed February 2, 2026,
<https://pmc.ncbi.nlm.nih.gov/articles/PMC8963868/>

38. Goal Babbling Permits Direct Learning of Inverse Kinematics - ResearchGate, accessed February 2, 2026,
https://www.researchgate.net/publication/224163311_Goal_Babbling_Permits_Direct_Learning_of_Inverse_Kinematics
39. Goal Babbling permits direct learning of inverse kinematics, accessed February 2, 2026, <https://www.honda-ri.de/pubs/pdf/1935.pdf>
40. Having multiple selves helps learning agents explore and adapt in complex changing worlds - PMC - PubMed Central, accessed February 2, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC10334746/>
41. Synthesising integrated robot behaviour through reinforcement learning for homeostasis, accessed February 2, 2026, <https://www.biorxiv.org/content/10.1101/2024.06.03.597087v2.full-text>
42. The Free Energy Principle for Perception and Action: A Deep Learning Perspective - NIH, accessed February 2, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC8871280/>
43. A Curiosity Algorithm for Robots Based on the Free Energy Principle - ResearchGate, accessed February 2, 2026, https://www.researchgate.net/publication/359073964_A_Curiosity_Algorithm_for_Robots_Based_on_the_Free_Energy_Principle
44. Deriving Motor Primitives Through Action Segmentation - PMC - PubMed Central, accessed February 2, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC3153847/>
45. Learning of Action Through Adaptive Combination of Motor Primitives - PMC, accessed February 2, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC2556237/>
46. Embodied Artificial Intelligence: Trends and Challenges - People, accessed February 2, 2026, <https://people.csail.mit.edu/iida/papers/PfeiferlidaEAIDags.pdf>
47. How to Install CUDA on Jetson Nano in 2025? - NVIDIA Developer Forums, accessed February 2, 2026, <https://forums.developer.nvidia.com/t/how-to-install-cuda-on-jetson-nano-in-2025/34204>
48. Leveraging the Power of GPUs with CuPy in Python - KDnuggets, accessed February 2, 2026, <https://www.kdnuggets.com/leveraging-the-power-of-gpus-with-cupy-in-python>
49. How to Build OPENCV with CUDA support on Jetson Nano or Xavier - YouTube, accessed February 2, 2026, <https://www.youtube.com/watch?v=WA7vjorifaQ>
50. Genesis, accessed February 2, 2026, <https://genesis-embodied-ai.github.io/>
51. Inverse Kinematics for Robotic Manipulators via Deep Neural Networks: Experiments and Results - MDPI, accessed February 2, 2026, <https://www.mdpi.com/2076-3417/15/13/7226>
52. tension: A Python package for FORCE learning - PMC, accessed February 2, 2026, <https://pmc.ncbi.nlm.nih.gov/articles/PMC9810194/>
53. OPENCV 4 + CUDA on Jetson Nano - YouTube, accessed February 2, 2026, https://www.youtube.com/watch?v=tFGZjVUR_Ck
54. Jupyter Notebook in Jetson Nano - NVIDIA Developer Forums, accessed February 2, 2026,

<https://forums.developer.nvidia.com/t/jupyter-notebook-in-jetson-nano/72586>

55. In Search for the Neural Mechanisms of Individual Development: Behavior-Driven Differential Hebbian Learning - Frontiers, accessed February 2, 2026,
<https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2015.00037/full>