Proyecto 2: Computación Nueromórfica

Ronald Duarte Barrantes , *Estudiante, ITCR* David Andrés Leitón-Flores,*Estudiante, ITCR,*

***Abstract*—Este informe introduce la computación neuromórfica como alternativa arquitectónica al cuello de botella memoria–cómputo de von Neumann en aplicaciones de borde. Se describen sus principios operativos (ejecución por eventos, paralelismo masivo, co-localización memoria–cómputo y aprendizaje local) y se contrastan dos líneas de implementación: aceleradores digitales dirigidos por eventos y enfoques analógicos in/near-memory basados en dispositivos no volátiles. Asimismo, se resumen los retos de programabilidad y la necesidad de co-diseño hardware–software. Como demostración, se presenta un caso de estudio de reconocimiento de expresiones faciales en tiempo real sobre hardware neuromórfico y se compara con aceleradores de borde convencionales mediante energía por inferencia, potencia, latencia, rendimiento y exactitud. Finalmente, se discuten implicaciones prácticas para decidir cuándo la neuromórfica ofrece ventajas medibles en sistemas embebidos de misión continua.**

***Index Terms*—Computación neuromórfica (Neuromorphic computing), Edge AI, In/near-memory computing, Redes neuronales de picos (Spiking neural networks, SNN).**

I. Introducción

Los avances recientes en aprendizaje profundo (deep learning) han expuesto con claridad el cuello de botella de la arquitectura de von Neumann: gran parte del costo en tiempo y energía proviene de mover datos entre memoria y cómputo, más que del cómputo mismo. Este desajuste se agrava en aplicaciones de borde, donde los dispositivos deben operar con presupuestos de potencia reducidos, latencias estrictas y, a menudo, con requisitos de privacidad que impiden enviar datos a la nube. En este contexto, la computación neuromórfica emerge como una alternativa arquitectónica inspirada en el cerebro humano, al combinar ejecución dirigida por eventos (spikes), paralelismo masivo, co-localización memoria–cómputo y aprendizaje local para lograr inferencia de muy baja energía y latencia [1, 2].

A alto nivel conviven dos familias tecnológicas: (i) aceleradores digitales por eventos que exponen neuronas y sinapsis programables y se interconectan mediante redes-en-chip ligeras; y (ii) enfoques analógicos in/near-memory que codifican pesos como conductancias físicas y realizan operaciones MAC en el propio sustrato, mitigando el tráfico memoria–cómputo [2, 3].

Este informe aborda, primero, qué es la computación neuromórfica y cómo se utiliza en la práctica (flujo, patrones de uso y métricas); luego presenta el panorama de implementaciones recientes (digital por eventos vs. in/near-memory analógico); y finalmente, muestra un caso de estudio de reconocimiento de expresiones faciales desplegado en hardware neuromórfico, comparado con aceleradores convencionales mediante energía por inferencia, potencia, latencia, rendimiento y exactitud. Además, se sintetizan desafíos de programabilidad y líneas de investigación en curso [1, 3, 2, 4, 5].

II. ¿Qué es la Computación Neuromórfica?

A. Definición y principios

La computación neuromórfica es un enfoque arquitectónico de hardware y software inspirado en el cerebro que busca reducir el costo energético y la latencia dominados por el movimiento de datos en arquitecturas de von Neumann. Se fundamenta en cuatro principios operativos: (i) *ejecución dirigida por eventos* (spikes) que activa cómputo sólo cuando hay información relevante; (ii) *paralelismo masivo y asíncrono* a través de redes de núcleos especializados; (iii) *co-localización memoria–cómputo* (*in/near-memory*) para minimizar tráfico de pesos/activaciones; y (iv) *aprendizaje local* cuando el sustrato lo permite [1, 2]. Estos rasgos habilitan inferencia de baja potencia y baja latencia en el borde cuando las cargas presentan esparsidad temporal (datos/eventos dispersos en el tiempo) [4]. Conceptualmente, el objetivo no es "replicar" el cerebro, sino capturar los mecanismos que lo hacen eficiente: actividad esparsa, comunicación por eventos y procesamiento distribuido [1].

B. Modelo computacional básico

Neuronas y sinapsis. El cómputo se organiza con modelos de neurona de disparo (p. ej., LIF/GLIF) caracterizados por umbral y fuga, y sinapsis con *peso* y *retardo* para modelar dinámica temporal; las reglas de aprendizaje pueden ser locales cuando están soportadas por la plataforma [1].

Comunicación por eventos. La información circula como trenes de picos (*spikes*) a través de una red-en-chip (NoC) que preserva la esparsidad temporal; en aceleradores digitales modernos, la mensajería por eventos coordina neuronas y núcleos manteniendo bajo el costo de comunicación [2].

Paradigmas de implementación. (a) *Digital por eventos* (p. ej., Loihi 2), que expone neuronas/sinapsis programables y redes-en-chip para escalar con eficiencia; (b) *analógico in/near-memory* (PCM (Phase-Change Memory)/RRAM (Resistive Random Access Memory) en *crossbars*), donde los pesos se codifican como conductancias físicas y las operaciones MAC se realizan "en la física" (leyes de Ohm/Kirchhoff), mitigando el cuello de botella memoria–cómputo y apuntando a latencias y energías por operación muy bajas [2, 3].

Como muestra la Fig. 1, Loihi organiza miles de neuronas programables por núcleo y las conecta mediante una NoC ligera que transporta spikes, reduciendo tráfico memoria–cómputo y activando cómputo sólo cuando hay eventos [2]

C. Diferencias con von Neumann

En la arquitectura clásica de von Neumann, el cómputo es *sincrónico* (gobernado por un reloj global) y el estado

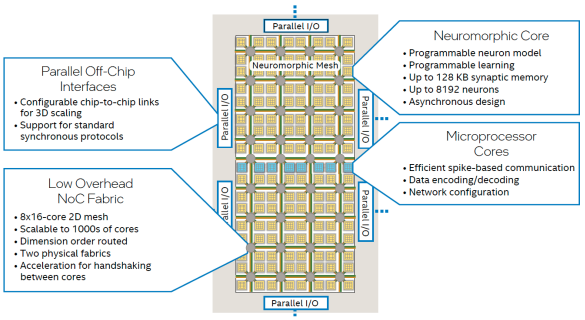


Fig. 1: Malla neuromórfica con núcleos spiking, NoC de baja sobrecarga y E/S paralela. Ilustra co-localización memoria–cómputo, ejecución por eventos y paralelismo masivo propios de arquitecturas neuromórficas [2].

se mantiene físicamente separado del procesador, de modo que el rendimiento y el consumo están fuertemente condicionados por el tráfico memoria–cómputo. La computación neuromórfica invierte estas premisas en cuatro ejes operativos: (i) **temporalidad por eventos**, donde el hardware sólo "trabaja" cuando hay información relevante (spikes), evitando despedicio de cómputo ; (ii) **co-localización de memoria y cómputo** (*in/near-memory*) que reduce movimientos de pesos/activaciones; (iii) **paralelismo masivo y asíncrono** a través de mallas de núcleos con mensajería ligera; y (iv) **representaciones y precisión adaptadas al tiempo**, priorizando energía/latencia sobre FLOPS [1, 2]. En plataformas digitales por eventos (p. ej., Loihi 2), los pesos se sitúan cerca del núcleo y la comunicación es por paquetes de eventos en una NoC; en sustratos analógicos in/near-memory (PCM/RRAM), los pesos son conductancias físicas y las operaciones MAC se realizan en el propio arreglo, mitigando el cuello de botella de von Neumann desde la física del dispositivo [2, 3]. Empíricamente, estas diferencias se traducen en reducciones sustanciales de energía por inferencia y potencia media en tareas por eventos del borde, manteniendo exactitud competitiva cuando el *pipeline* está adaptado al régimen espaciotemporal del hardware [4]. Es importante notar que estas diferencias son *arquitectónicas* y no limitan la capacidad de cómputo: existen resultados formales que establecen la Turing-completitud del paradigma neuromórfico [5].

En von Neumann se optimiza una *tubería de instrucciones* sincronizada y un subsistema de memoria jerárquico; en neuromórfica se optimiza un *flujo de eventos* con estado local y comunicación esparsa. Por ello, también cambian las *métricas foco*: de FLOPS y ancho de banda de memoria a energía por inferencia, potencia y latencia baja [1, 4].

La figura 2 sintetiza el tránsito desde cómputo *sincrónico y denso* (uso del clock y con fuerte separación memoria–cómputo) hacia cómputo *asíncrono y por eventos* con *sparsity* explotable. En el paradigma convencional/paralelo, el rendimiento está dominado por el ancho de banda de memoria y los FLOPS; en neuromórfica, el foco operativo migra a *energía por inferencia* y *latencia end-to-end*, coherente con los principios de ejecución por eventos, paralelismo masivo y co-localización memoria–cómputo. Esta figura es el puente

TABLE I: Contraste operativo: von Neumann vs. neuromórfica

Eje	**von Neumann**	**Neuromórfica**
Temporalidad	Sincrónica (reloj continuo)	Por eventos (actividad esparsa)
Memoria–cómputo	Separados (tráfico dominante)	Co-localizados (in/near-memory)
Paralelismo	ILP/DLP, SIMD/CMP	Many-core asíncrono, SNN
Comunicación	Buses/caches	NoC con paquetes de spikes
Representación	Densa, FP/INT	Temporal, baja precisión útil
Métricas foco	FLOPS, BW memoria	Energía/inf., mW, ms
Soporte HW	ALU+jerarquía caché	Núcleos spiking, crossbars PCM/RRAM

visual con la comparativa formal de la tabla I.

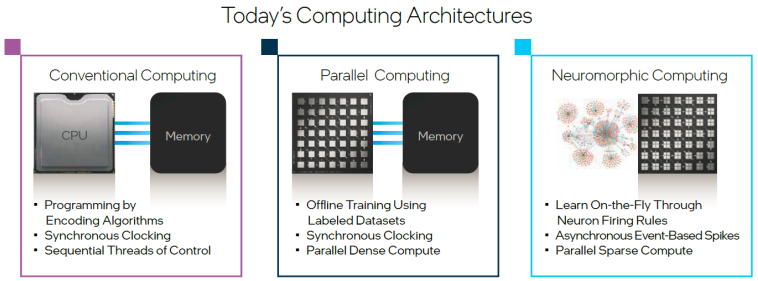


Fig. 2: Arquitecturas actuales: convencional y paralela (sincronía y cómputo denso) vs. neuromórfica (eventos asíncronos y cómputo disperso). Refuerza el cambio de foco: de FLOPS/BW a energía por inferencia y latencia. [2]

III. Importancia y aplicaciones

La computación neuromórfica aporta eficiencia energética y latencias competitivas en cargas con esparsidad temporal, al activar cómputo sólo ante eventos y minimizar el tráfico memoria–cómputo mediante co-localización del estado [1, 2]. En el borde, esto se traduce en *energía por inferencia* y *potencia media* sensiblemente menores frente a aceleradores convencionales, manteniendo exactitud cuando el *pipeline* respeta la temporalidad del hardware [4]. Además, al ejecutar *on-device*, mejora la privacidad y reduce dependencia de la nube.

Ámbitos de uso típicos.

- **Visión por eventos (DVS):** detección/clasificación con *always-on* de muy baja potencia (aprovechando la esparsidad del sensor y del procesamiento).
- **Audio/Keyword Spotting:** despertar por palabra clave con latencias de decenas de ms y operación sostenida en bajo consumo.
- **Robótica y control**: la ejecución por eventos favorece respuesta determinista y estabilidad en lazo.
- **IoT industrial**: detección de anomalías y conteo con telemetría energética baja y procesamiento local.

En síntesis, cuando el flujo de datos es esparso y el presupuesto de potencia es crítico, la neuromórfica ofrece ventajas medi-

bles sin sacrificar la precisión requerida por la aplicación [2, 4].

IV. Desafíos y Consideraciones en Computación Neuromórfica

A. Problemas Fundamentales en la Programación Neuromórfica

La programación de sistemas neuromórficos enfrenta varios desafíos críticos que surgen de sus diferencias fundamentales con la computación clásica:

- **Incompatibilidad con Paradigmas Tradicionales**: No es posible aplicar la teoría de ciencia computacional convencional para entender los procesos computacionales en sistemas neuromórficos debido a diferencias fundamentales en dominio temporal, plasticidad, estocasticidad, descentralización e inobservabilidad [1].
- **Falta de Abstracciones Universales**: Actualmente no existen métodos universalmente aceptados para la programación neuromórfica. No hay un lenguaje de programación neuromórfico estándar, y no está claro si requerirá nueva sintaxis o representaciones visuales [1].
- **Dificultades de Observabilidad**: Los sistemas neuromórficos, especialmente los analógicos y de señal mixta, frecuentemente muestran observabilidad limitada, donde el estado del sistema solo puede leerse parcialmente. Esto representa una dificultad clave para cómputos plásticos que cambian con el tiempo [1].
- **Problemas de Plasticidad y Cambio Dinámico**: Cuando el hardware cambia durante la ejecución del programa, se requiere que la máquina sea insensible al cambio o que el modelo computacional incluya el cambio para mantener la corrección del cómputo [1].

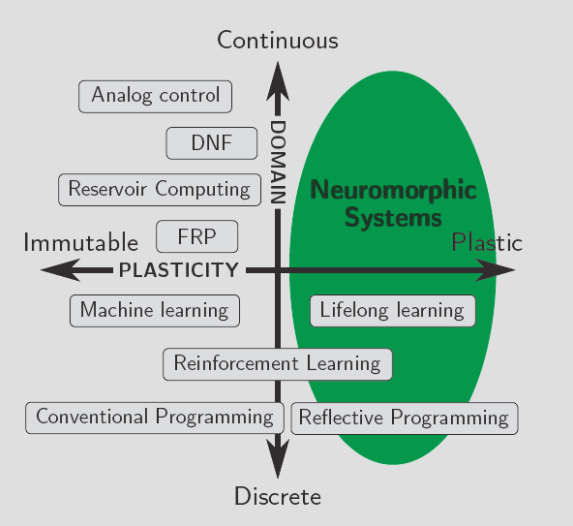


Fig. 3: Modelos de programación y computación graficados según su dominio operativo (continuo vs. discreto) y la maleabilidad del cómputo durante la ejecución (inmutable vs. plástico).

B. Metodologías de Programación Emergentes

1) Enfoques Basados en Aprendizaje Automático:

- **Conversión ANN-to-SNN**: Es posible pero típicamente no óptima porque las SNNs resultantes no aprovechan el poder computacional de las neuronas espikeantes, limitando su dinámica más rica al dominio menos expresivo de las ANNs [1].
- **Entrenamiento Offline**: Métodos como backpropagation pueden implementarse directamente en SNNs usando gradientes sustitutos, pero el re-entrenamiento frecuente crea una sobrecarga grande que limita el rendimiento y aplicabilidad [1].
- **Compilación Neuromórfica**: Se propone como un framework general para compilar aproximadamente redes neuronales en diferentes sistemas de hardware, adaptándose automáticamente a restricciones físicas [1].

2) Aprendizaje en el Dispositivo (Online Learning): Los métodos de aprendizaje en el dispositivo son un tema activo de investigación para evitar la sobrecarga del re-entrenamiento frecuente [1]:

- **Plasticidad**: Paradigma popular donde reglas de aprendizaje locales modifican la conectividad (plasticidad estructural) y fuerzas de conexión (plasticidad sináptica) de una SNN.
- **Búsqueda Evolutiva y Meta-Aprendizaje**: Se utilizan para (re)descubrir reglas de plasticidad deseables, ya que no está claro qué reglas locales producirán un comportamiento a nivel de red particular [1].

3) Métodos Evolutivos:

- **Ventaja**: Pueden optimizar conjuntamente la arquitectura y pesos de la red, diseñando y entrenando la red simultáneamente sin requerir que la red sea diferenciable.
- **Desventaja**: Pueden converger más lentamente que otros métodos de entrenamiento y las arquitecturas resultantes no son fácilmente interpretables o reutilizables para diferentes tareas.

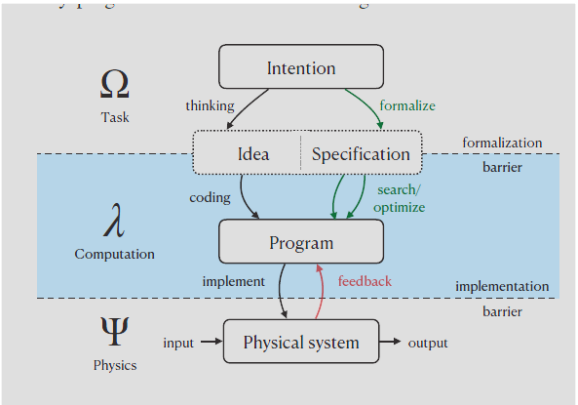


Fig. 4: La intención abstracta vive en el espacio informal Ω de todas las posibles tareas computacionales.

Es un esquema representativo del funcionamiento de una red neuromórfica, que muestra el proceso esperado según la tarea a implementar y cómo esta se va modificando hasta lograr el

objetivo esperado. Por ello se observa una realimentación entre la parte del programa y la implementación física del sistema.

C. Plataformas y Herramientas de Programación

1) Abstracciones de Alto Nivel:

- **Neural Engineering Framework (NEF)**: Permite que sistemas dinámicos sean automáticamente destilados en redes de neuronas espikeantes usando el entorno de programación Nengo.
- **Lava**: Framework de programación neuromórfica de código abierto que incluye librerías de algoritmos neuromórficos para optimización, redes de atractores, métodos de deep learning para SNNs, y VSAs.
- **Fugu**: Mecanismo independiente del hardware para componer SNAs. Un programa se especifica como un grafo computacional, reminiscente de programación de flujo de datos [**1**].

2) Representaciones Intermedias:

- **Neuromorphic Intermediate Representation (NIR)**: Provee una abstracción para cómputos de tiempo continuo con estructura de grafo, que puede soportar una variedad más amplia de paradigmas de programación neuromórfica.
- **Integración Heterogénea**: NIR integra representaciones heterogéneas como el Neural Engineering Framework, Lava, Fugu, PyNN, y NeuroML, permitiendo que múltiples representaciones y paradigmas cooperen a través de una representación compartida.

D. Métricas y Compromisos de Diseño

Los sistemas neuromórficos presentan varios compromisos fundamentales que deben considerarse en su diseño e implementación:

- **Precisión vs. Energía**: La naturaleza de baja precisión y procesamiento basado en eventos ofrece eficiencia energética pero puede comprometer la exactitud computacional [1].
- **Plasticidad vs. Estabilidad**: La capacidad de cambiar durante la ejecución permite adaptabilidad pero introduce desafíos para mantener la estabilidad computacional [1].
- **Descentralización vs. Control**: El procesamiento distribuido ofrece robustez pero dificulta el control global y la observabilidad del sistema [1].
- **Tiempo Continuo vs. Sincronización**: La operación en tiempo real permite reactividad pero complica la sincronización y coordinación de procesos [1].

E. Direcciones Futuras Críticas

Para que los sistemas neuromórficos escalen a sistemas de computación grandes y heterogéneos, se requieren modelos computacionales comúnmente aceptados [1]. Los enfoques modernos deben:

- Acomodar la naturaleza simultáneamente analógica y digital de la computación neural
- Capturar el cambio inherente en el sustrato subyacente
- Modelar ruido a nivel de señal y mantenerse robusto a perturbaciones
- Permitir procesamiento de información basado en eventos
- Depender únicamente de información localmente disponible [1]

V. Implementaciones y tecnologías actuales

A. Aceleradores digitales por eventos (Loihi 2)

A nivel digital, Loihi 2 integra núcleos neuromórficos interconectados por una NoC que enruta spikes (y, cuando se requiere, payload) preservando la esparsidad temporal. El chip expone configurabilidad de neuronas y sinapsis, además de soporte para aprendizaje local, habilitando paralelismo masivo y baja energía por inferencia en cargas por eventos .Su desempeño práctico frente a aceleradores edge convencionales se ilustra en el caso de estudio de este informe.

B. Sistemas analógicos in/near-memory (PCM/RRAM/crossbar)

En sustratos analógicos, los pesos se codifican como conductancias de dispositivos no volátiles (PCM/RRAM) dispuestos en crossbars. Las operaciones MAC se realizan “en la física” (leyes de Ohm y Kirchhoff), lo que reduce el movimiento de datos y mejora la eficiencia. Este enfoque requiere co-diseño HW–SW para lidiar con variabilidad, endurance y límites del entrenamiento on-chip, pero ofrece una vía clara para mitigar el cuello de botella de von Neumann desde el dispositivo [1].

C. Comparación sintética

En la práctica, Loihi 2 (digital por eventos) ofrece neuronas/sinapsis programables, aprendizaje local y escalado sobre NoC, logrando latencias de milisegundos con energía por inferencia muy baja en *edge AI* esparso; los enfoques analógicos in/near-memory (PCM/RRAM) ejecutan MAC directamente en el arreglo para minimizar tráfico memoria–cómputo, pero hoy suelen entrenarse off-chip y requieren mitigaciones de variabilidad [1]. Ambos comparten el objetivo de **romper el cuello de botella memoria–cómputo** desde la arquitectura (digital/eventos) o desde el dispositivo (analógico).

VI. Demostración de Turing-Completitud en Computación Neuromórfica

La demostración formal de que la computación neuromórfica es Turing-completa constituye un hito fundamental en la teoría de la computación, estableciendo las bases teóricas para su aplicación como paradigma de propósito general. Esta sección detalla el proceso de demostración y sus implicaciones prácticas.

A. Marco Teórico y Fundamentos

La prueba se sustenta en la equivalencia entre la Máquina de Turing y las funciones μ -recursivas, donde un modelo computacional se considera Turing-completo si puede calcular el mismo conjunto de funciones que una Máquina de Turing. Como se establece en la sección de trabajos relacionados, el conjunto de funciones μ -recursivas representa precisamente este conjunto, proporcionando la base formal para la demostración. [5]

B. Metodología de Demostración

La estrategia de prueba sigue un enfoque constructivo mediante la implementación de los seis componentes fundamentales de las funciones μ -recursivas:

1) Funciones Base Implementadas:

- **Función Constante (C_k)**: Circuito neuromórfico que ignora la entrada y genera siempre el valor constante k mediante el uso estratégico de pesos sinápticos cero y valores umbral apropiados.
- **Función Sucesor (S)**: Implementación que incrementa la entrada en una unidad mediante la combinación de la señal de entrada con un valor constante, demostrando capacidades aritméticas básicas.
- **Función de Proyección (P)**: Circuito complejo que actúa como selector de índice, utilizando mecanismos de inhibición y comparación para seleccionar elementos específicos de un conjunto de entradas.

2) Operadores Fundamentales:

- **Operador de Composición ($\circ$)**: Permite la conexión modular de circuitos, donde las salidas de múltiples funciones se convierten en entradas para otras funciones, estableciendo las bases para la construcción de programas complejos.
- **Operador de Recursión Primitiva (ρ)**: Implementa recursión mediante un circuito que combina un caso base (i=0) y un paso recursivo (i>0), utilizando mecanismos de memoria temporal y control iterativo.
- **Operador de Minimización (μ)**: Representa el operador más complejo, capaz de realizar búsquedas no acotadas mediante la evaluación iterativa de funciones hasta encontrar el mínimo que satisface una condición específica.

C. Análisis de Correctitud

La demostración de correctitud se basa en los siguientes aspectos clave:

- **Completitud**: Cada circuito implementa exactamente la semántica de la función -recursiva correspondiente, verificada mediante el análisis del comportamiento de spikes y estados internos.[5]
- **Terminación**: Para operadores recursivos, se garantiza la terminación mediante condiciones de parada bien definidas y mecanismos de control temporal.
- **Consistencia**: Los circuitos mantienen coherencia en la representación de datos a través de la codificación espaciotemporal de spikes, asegurando la preservación de la información durante el cómputo.

D. Implicaciones Teóricas

La demostración exitosa establece que:

- **Universalidad Computacional**: Los sistemas neuromórficos poseen el mismo poder computacional que las máquinas de Turing convencionales, capaz de implementar cualquier algoritmo computable.
- **Independencia de Plasticidad**: A diferencia de demostraciones anteriores, esta prueba no requiere plasticidad sináptica, estableciendo la Turing-completitud para modelos neuromórficos minimalistas.
- **Generalización**: El modelo de dos parámetros por neurona/sinapsis es suficiente para la computación universal, proporcionando un marco minimalista pero expresivo.

E. Verificación Experimental

Si bien la demostración es principalmente teórica, se incluyen consideraciones para verificación práctica:

- **Simulación en Máquinas von Neumann**: La capacidad de simular exactamente computaciones neuromórficas en simuladores como NEST proporciona un mecanismo de validación experimental.
- **Análisis de Complejidad**: El framework propuesto permite evaluar la eficiencia de los circuitos en términos de espacio (número de neuronas/sinapsis) y tiempo (longitud de rutas sinápticas).
- **Modularidad**: La representación de circuitos complejos mediante cajas abstractas facilita la composición y verificación de sistemas a gran escala.

F. Limitaciones y Consideraciones Prácticas

Aunque la demostración establece la Turing-completitud teórica, existen consideraciones prácticas importantes:

- **Eficiencia en Recursos**: Operadores complejos como minimización requieren mayor número de neuronas y presentan latencias más altas.
- **Precisión Numérica**: La implementación utiliza valores enteros, mientras que aplicaciones prácticas pueden requerir representaciones en punto flotante.
- **Escalabilidad**: Si bien los circuitos son teóricamente correctos, la implementación en hardware físico introduce consideraciones adicionales de sincronización y ruido.

G. Conclusiones de la Demostración

La prueba de Turing-completitud neuromórfica establece que:

1) Los sistemas neuromórficos son capaces de computación universal sin necesidad de componentes adicionales beyond neuronas y sinapsis básicas.[5]
2) El paradigma neuromórfico mantiene su ventaja en eficiencia energética mientras preserva la completitud computacional.
3) Se proporciona una base teórica sólida para el desarrollo de compiladores y herramientas de programación neuromórfica.
4) Se abre el camino para aplicaciones de propósito general en dominios donde la eficiencia energética es crítica. [5]

Esta demostración posiciona la computación neuromórfica como un paradigma viable no solo para aplicaciones especializadas de machine learning, sino como alternativa general a las arquitecturas convencionales en la era post-Moore.

VII. Comparación con arquitecturas de propósito general

En neuromórfica, el "ISA" efectivo se aproxima a un *microcódigo neuronal* basado en eventos; la jerarquía de memoria migra a esquemas *near-/in-memory*; el paralelismo se materializa como ejecución masiva, asíncrona y esparsa; el control de flujo se modela mediante colas y *backpressure*; y la interconexión es una malla NoC optimizada para paquetes de *spikes* [1, 2]. En métricas, el foco pasa de FLOPS y ancho de banda de memoria a *energía por inferencia*, *potencia* y *latencia end-to-end*, relevantes en sistemas embebidos de misión continua [4].

En línea con el contraste de la tabla I, la figura 5 muestra extractos del conjunto de instrucciones de *Loihi 2* que materializan un "microcódigo neuronal" basado en estado local y eventos: (i) **RMW/RDC** permite leer–modificar–escribir variables del estado neuronal en memoria local; (ii) **MOV/SEL** mueve/selecciona parámetros entre registros y ese espacio local; (iii) primitivas lógicas y aritméticas (**AND/OR/SHL**, **ADD/NEG/MIN**, **MUL_SHR**) soportan cálculo de baja precisión útil; (iv) **LT/GE/EQ** con **SKP_C/JMP_C** habilitan control de flujo reactivo; y (v) **SPIKE/PROBE** genera/sondea eventos como mecanismo de comunicación. Este ISA ejemplifica la co-localización memoria–cómputo y la mensajería por eventos descritas en el informe: el programa opera sobre *estado cercano* y se comunica con la NoC mediante *spikes*, en contraste con el modelo clásico ALU+jerarquía de caché de von Neumann [2].

Table 1. Highlights of the Loihi 2 Instruction Set

OP CODES	DESCRIPTION
RMW, RDC *read-modify-write, read-and-clear*	Access neural state variables in the neuron's local memory space
MOV, SEL *move, move if 'c' flag*	Copy neuron variables and parameters between registers and the neuron's local memory space
AND, OR, SHL *and, or, shift left*	Bitwise operations
ADD, NEG, MIN *add, negate, minimum*	Basic arithmetic operations
MUL_SHR *multiply shift right*	Fixed precision multiplication
LT, GE, EQ *less than, not equal, equals*	Compare and write result to 'c' flag
SKP_C, JMP_C *skip ops, jump to program address based on 'c' flag*	Branching to navigate program
SPIKE, PROBE *spike, send probe data*	Generate spike or send probe data to processor

Fig. 5: Extracto del ISA de Loihi 2: acceso a variables neuronales locales (RMW/RDC), movimiento/selección (MOV/SEL), control de flujo (SKP_C/JMP_C) y primitivas de evento (SPIKE/PROBE). Ejemplifica un "microcódigo neuronal" orientado a cómputo por eventos. [2]

VIII. Caso de estudio: reconocimiento de expresiones faciales en tiempo real

A. Requisitos del sistema

El sistema debe cumplir con especificaciones críticas para operar en aplicaciones prácticas. En términos de potencia, el consumo debe ser mínimo, idealmente en el rango de milivatios (mW) para permitir operación prolongada con batería. La latencia objetivo debe garantizar al menos 20-30 FPS (cuadros por segundo), equivalente a 33-50 ms por inferencia, para considerarse tiempo real. La exactitud mínima requerida supera el 95% en el dataset CK+ con siete expresiones faciales. En memoria, los modelos deben adaptarse a las limitaciones de los dispositivos edge, mientras que la privacidad se asegura mediante procesamiento local sin transmisión a la nube.[4]

B. Metodología

El proceso sigue un pipeline bien definido. La etapa de **preproceso** incluye detección de bordes para aumentar la esparsidad de las imágenes. La **codificación** convierte imágenes estáticas en trenes de picos mediante una capa convolucional especial. La conversión **ANN→SNN** involucra: sustitución de funciones ReLU por neuronas LIF (Leaky-Integrate-and-Fire), reemplazo de capas de pooling por convoluciones con stride, y re-entrenamiento para adaptar los pesos a la dinámica de picos. Todo el proceso se realiza mediante herramientas de conversión automática sin entrenamiento spiking directo.

C. Plataforma y mapeo

La implementación utiliza el chip neuromórfico **Intel Loihi**, que consta de 128 núcleos con capacidad para 1,024 neuronas cada uno. El **particionado** del modelo requirió dividir las capas convolucionales en bloques que no excedieran el límite de neuronas por núcleo, utilizando 91 de los 128 núcleos disponibles. Las interfaces de **E/S** manejan la comunicación entre la codificación off-chip y el procesamiento on-chip. La **toolchain** NengoDL facilitó la conversión y despliegue del modelo.

D. Métricas y protocolo de medición

Las mediciones incluyen: **latencia end-to-end** desde la entrada hasta la clasificación, **potencia media** durante inferencia, **energía por inferencia** calculada como potencia × latencia, **FPS** derivado de la latencia, y **exactitud** en el dataset CK+. Para los aceleradores edge se utilizó un multímetro USB, mientras que para Loihi se emplearon sensores hardware integrados. El dataset consistió en imágenes de 48×48 píxeles en escala de grises, con y sin preprocesamiento de detección de bordes.

E. Resultados y análisis de trade-offs

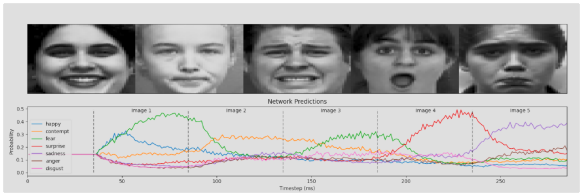
El análisis revela importantes trade-offs: mientras los aceleradores edge ofrecen menor latencia y mayor FPS, **Loihi consume 4.8× menos energía** que el Coral Dev Board, el más eficiente entre los aceleradores tradicionales. En el

TABLE II: Comparativa de plataformas

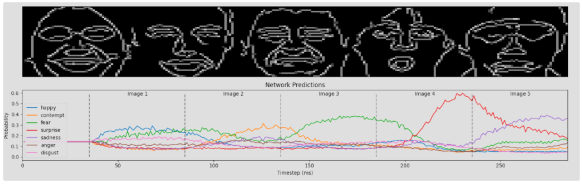
2*Dispositivo	Prec. (%)	Lat. (ms)	2*FPS	En. (mJ)
Raspberry Pi	96.95	2.88	347	4.49
Jetson Nano	97.46	1.62	617	1.47
Coral Dev	96.25	0.39	2564	0.203
Loihi	**97.40**	**35.0**	**28.5**	**0.042**

balance precisión vs energía, Loihi mantiene la mayor precisión (97.40%) con el menor consumo energético. Respecto a latencia vs esparsidad, el preprocesamiento con detección de bordes reduce la potencia dinámica en 50% y mejora la precisión, demostrando que mayor esparsidad compensa la mayor latencia inherente al hardware neuromórfico.[4]

Representación gráfica del rendimiento de un sistema neuromórfico en la detección de expresiones faciales. Se observa un incremento significativo en la probabilidad de detección cuando el sistema procesa expresiones específicas, demostrando la efectividad del enfoque neuromórfico para esta tarea



(a)



(b)

Fig. 6: Resultados de inferencia de cinco imágenes probadas en Loihi: (a) imagen en escala de grises, y (b) imágenes con detección de bordes.

IX. Conclusiones

- La computación neuromórfica aborda el cuello de botella memoria–cómputo de von Neumann mediante ejecución por eventos, paralelismo masivo y co-localización memoria–cómputo, logrando inferencia de baja energía y latencia en flujos esparsos del borde.
- Existen dos familias complementarias: (i) digital por eventos (p. ej., Loihi 2), con alta programabilidad y aprendizaje local; y (ii) analógica in/near-memory (PCM/RRAM), que "calcula en la física" pero exige mitigar variabilidad y suele entrenarse *off-chip*. Ambas minimizan el movimiento de datos.
- Las métricas relevantes para el borde son energía por inferencia, potencia y latencia *end-to-end* más que FLOPS. El caso de estudio muestra ventajas claras en energía (aunque no siempre en latencia) y que aumentar la esparsidad del *pipeline* mejora eficiencia y puede sostener la exactitud.
- La neuromórfica es capaz (Turing-completa), pero su límite actual es la programabilidad: faltan abstracciones/IR estándar y aprendizaje en-dispositivo robusto. Es especialmente atractiva para *always-on* y tareas esparsas con presupuestos de mW; prioridades futuras: co-diseño HW–SW, observabilidad y control de variabilidad analógica.

X. Referencias

References

[1] S. Abreu and J. E. Pedersen. Neuromorphic programming: emerging directions for brain-inspired hardware. In *Proc. International Conference on Neuromorphic Systems (ICONS)*, 2024.

[2] Intel Labs. Taking neuromorphic computing to the next level with loihi 2. *Intel Labs Research Brief*, 2021.

[3] D. C. Crowder et al. Ai-enhanced codesign of neuromorphic circuits. In *Proc. IEEE Midwest Symposium on Circuits and Systems (MWSCAS)*, 2023.

[4] H. Smith et al. Realtime facial expression recognition: neuromorphic hardware vs. edge ai accelerators. *arXiv preprint*, 2024. arXiv:2403.08792.

[5] P. Date et al. Neuromorphic computing is turing-complete. In *Proc. International Conference on Neuromorphic Systems (ICONS)*, 2022.