

Implementació Compressor Basat en Xarxes Neurals per a Ús en Satèl·lit

Portant Deep Learning a l'Espai amb Recursos Limitats

Cristhian Omar Añez López

Febrer 2026

Escola d'Enginyeria (EE)

Universitat Autònoma de Barcelona (UAB)

Tutor: Sebastià Mijares Verdú

Contingut de la Presentació

Introducció i Motivació

Estat de l'Art

Metodologia i Implementació

Resultats Experimentals

Conclusions

Introducció i Motivació

El Problema: Massa Dades, Poc Ample de Banda

Context actual de l'observació terrestre:

- Missions com **Sentinel-2**, EnMAP, PRISMA generen terabytes diaris
- Imatges multiespectrals/hiperespectrals d'alta resolució
- L'ample de banda de descàrrega és limitat

En CubeSats i nanosatèl·lits:

- Potència molt limitada (watts)
- Poca memòria RAM (≤ 1 GB)
- CPU de baix consum (ARM)

→ Cal comprimir les dades a bord!

El Desafiament

Com executar algorismes intel·ligents de compressió en maquinari molt limitat?

Objectius del TFG

Objectiu General

Desenvolupar i validar una versió eficient del compressor **SORTENY** orientada a plataformes amb recursos limitats.

▷ Implementació

- Reimplementar SORTENY en **C pur**
- Eliminar dependències de TensorFlow
- Arquitectura portable per ARM

⚙️ Optimització

- Reduir consum de memòria
- Explotar paral·lelisme (OpenMP)
- Vectorització SIMD (NEON)

☒ Avaluació

- Comparar amb Python/TensorFlow
- Benchmark vs CCSDS-122
- Validar qualitat de reconstrucció

🔑 Viabilitat

- Provar en Raspberry Pi 3B+
- Simular restriccions de CubeSat
- Demostrar factibilitat real

Estat de l'Art

Compressió Espacial: Mètodes Clàssics (CCSDS)

CCSDS 122.0-B-1: L'Estàndard Industrial

1. Transformada Wavelet (DWT)

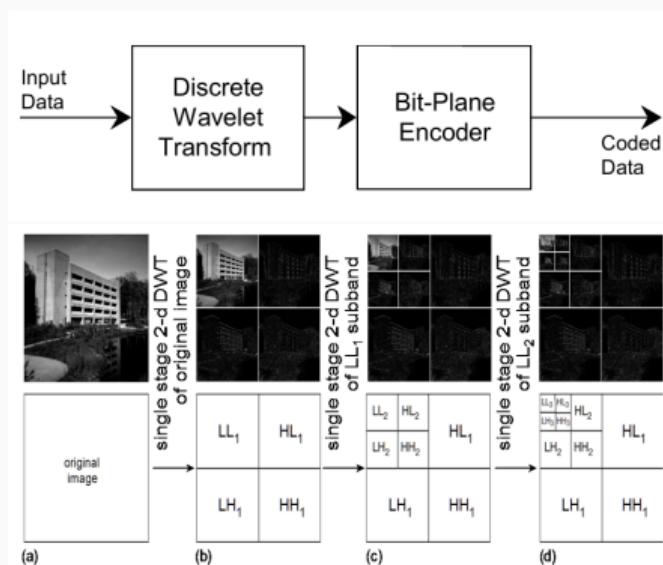
- Descomposició en sub-bandes
- Filtres 5/3 (lossless) o 9/7 (lossy)
- Compactació d'energia

2. Codificador de Plans de Bits (BPE)

- Codificació progressiva MSB→LSB
- Estructura d'arbres (Zerotrees)
- Flux escalable

✓ Ràpid, eficient, provat en vol

✗ Transformades **fixes**, no adaptatives



Compressió Apresa: El Model SORTENY

SORTENY: Spectral Orthogonal Transform Encoder

Desenvolupat per l'IEEC, combina:

1. Transformada Espectral Apresa

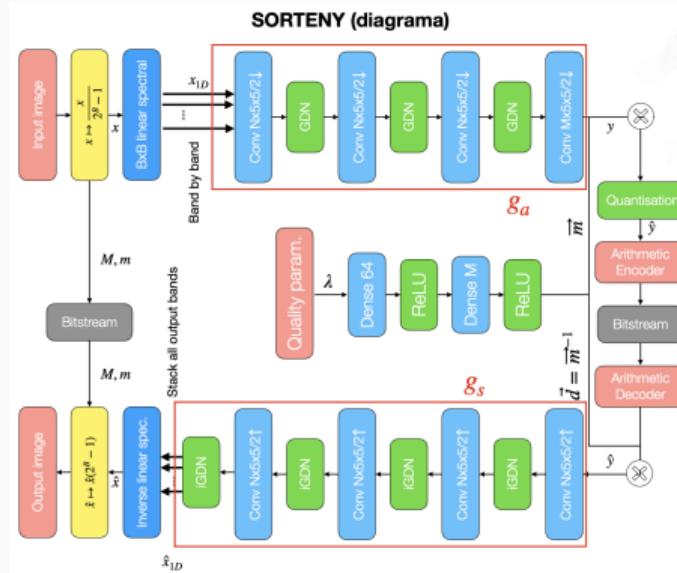
- Similar a PCA/KLT
- Descorrelació de bandes

2. CNN per Compressió Espacial

- Convolucions amb stride
- Normalització GDN

3. Modulació de Qualitat (λ)

- Taxa variable sense reentrenar
- Un únic model per tot



Avantatge Clau

Adaptabilitat: Les transformades s'aprenen de les dades reals, optimitzant qualitat per bit.

El Problema de SORTENY

Requisits de la Versió Python/TensorFlow

- **RAM:** 800 MB només per carregar el model
- **Dependències:** TensorFlow, NumPy, Python runtime
- **Pes:** Gigabytes d'instal·lació

La Pregunta Clau

És viable executar un compressor basat en xarxes neuronals en un entorn embarcat realista?

En una Raspberry Pi 3B+:

- 1 GB RAM total (compartida amb GPU)
- El sistema col·lapsa (OOM Kill)
- Swap massiu → rendiment catastròfic



Sí, però com?

Metodologia i Implementació

Estratègia: Reimplementació Nativa en C

Fase 1: Extracció de Pesos

- Serialitzar tensors a binaris plans
- Generar índex de metadades

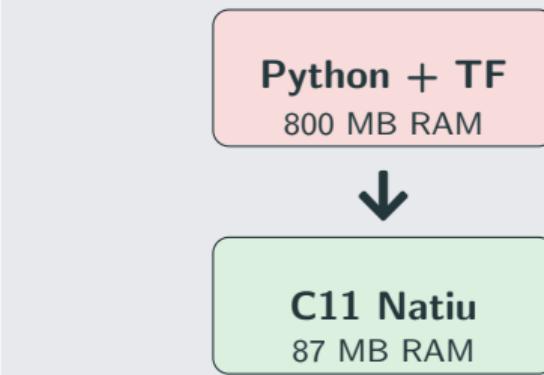
Fase 2: Motor d'Inferència

- **Convolució 2D:** Padding SAME manual
- **GDN:** Normalització divisiva
- **Transformada Espectral:** Producte matricial

Resultat:

- ✓ Zero dependències
- ✓ Executable únic i portable

Desacoblament del Framework



Compilació Optimitzada

```
gcc -O3 -mcpu=cortex-a53  
-mfpu=neon-vfpv4 -fopenmp
```

Gestió de Memòria: Estratègia “Ping-Pong”

Problema Original:

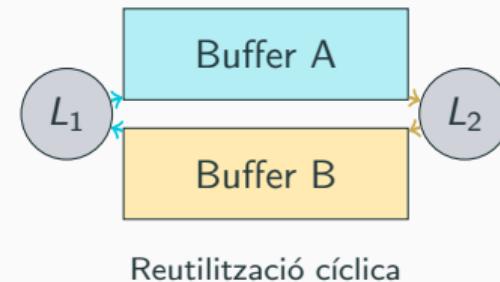
- Cada capa crea nous tensors
- Memòria creix linealment $O(N)$
- Pics de 800+ MB

Solució Ping-Pong:

- Només 2 buffers de treball
- Alternança: $A \rightarrow B \rightarrow A \rightarrow B$
- Memòria constant $O(1)$

Fórmula de consum:

$$\text{Mem}_{\text{total}} \approx \text{Pesos} + \max_i(\text{Dim Capa}_i) \times 2$$



Resultat

Reducció de memòria:
800 MB → 87 MB
(9.2x menys)

Optimització: Paral·lelisme i Vectorització

ParallelGroupisme de Fils (OpenMP)

- Independència en canals de sortida
- Distribució en 4 nuclis Cortex-A53
- Directiva: #pragma omp parallel for

Vectorització SIMD (NEON)

- Unitats vectorials de 128 bits
- Processar múltiples píxels per cicle
- Flags: -mcpu=cortex-a53 -mfpu=neon

Validació de Paritat Numèrica

Errors crítics detectats i corregits:

1. Normalització d'entrada ($\div 65535$)
2. Fórmula GDN incorrecta
3. Escalat de λ

Resultat Final

99.98% dels latents
són **bit a bit idèntics**
entre C i Python

Resultats Experimentals

Plataforma de Proves

- **Hardware:** Raspberry Pi 3B+
- **CPU:** ARM Cortex-A53 @ 1.4 GHz
- **RAM:** 1 GB LPDDR2
- **OS:** Raspberry Pi OS 64-bit

Dataset

- Imatge Sentinel-2 (T31TCG)
- Dimensions: 512×512 px
- 8 bandes espectrals
- 16 bits/píxel (4.19 MB total)

Comparatives

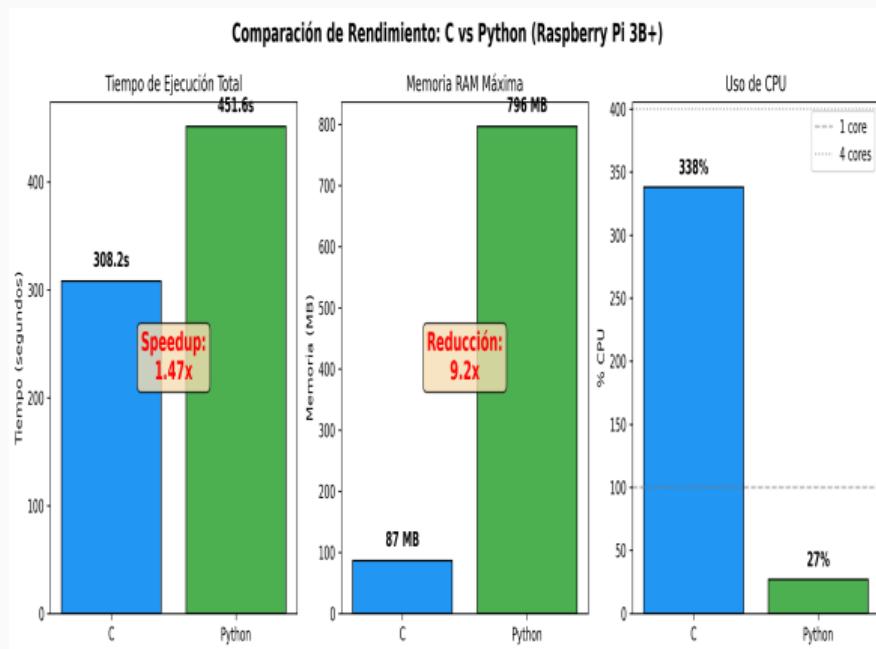
Mètode	Detalls
SORTENY C	GCC 10.2, OpenMP
SORTENY Py	Python 3.9, TF 2.16
CCSDS 122	MHDC (UAB)

Paràmetre de Qualitat

$$\lambda = 0.1$$

(Alta qualitat)

Resultats: Rendiment i Memòria



Anàlisi:

- **C:** Satura els 4 nuclis (338% CPU)
- **Python:** 75% del temps en swap/IO
- La memòria era el **coll d'ampolla real**

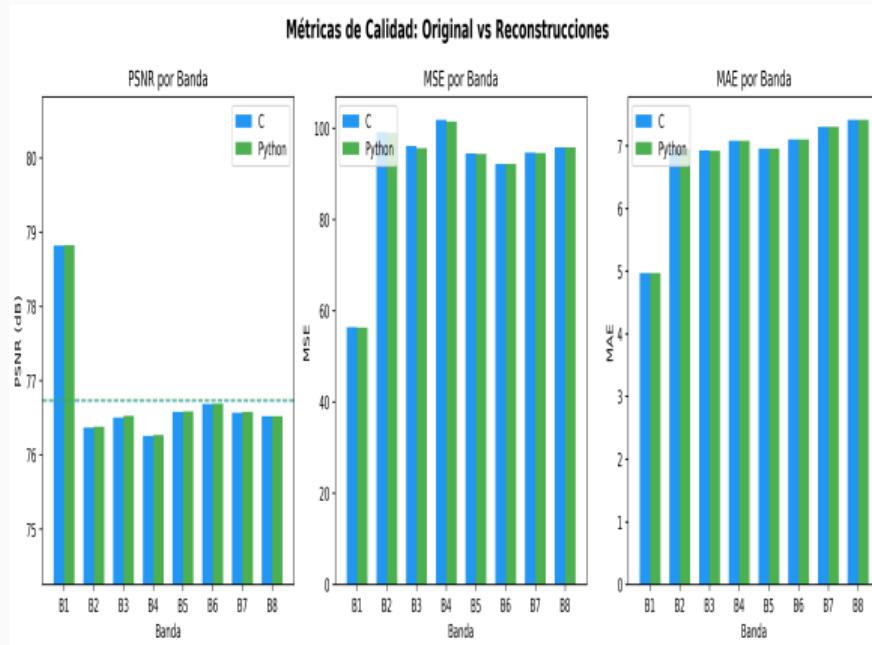
Xifres Clau

Speedup: 1.47x

RAM: 796 → 87 MB (9.2x)

CPU: 338% vs 27%

Resultats: Qualitat de Reconstrucció



Anàlisi:

- Les dues línies (C i Python) se superposen perfectament
- Variació natural per contingut espectral
- Diferències < 0.03 dB

Conclusió

PSNR Global: 76.73 dB
Qualitat **idèntica**
entre C i Python

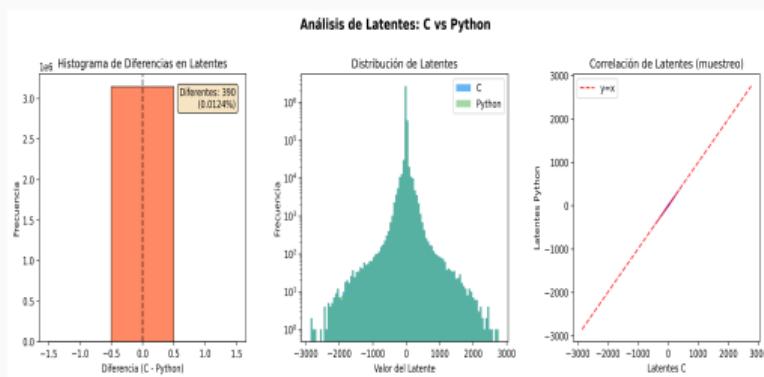
Validació Visual i Paritat de Latents

Paritat Numèrica:

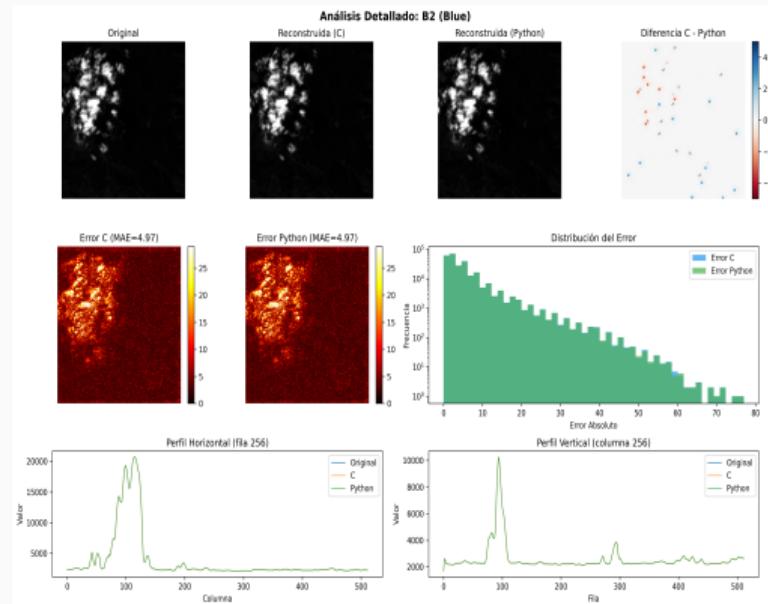
- 3.15 milions de valors latents
- **99.9876%** són idèntics
- Només 390 amb diferència ± 1

Causa de les diferències:

- Precisió FP: NEON vs AVX/SSE
- Arrodoniment bancari en $\times 5$



Inspecció Visual:



Detall Banda 1: Original vs C vs Python
Imatges visualment indistingibles

Comparativa amb CCSDS-122 (Benchmark Industrial)

Taula 1: SORTENY ($\lambda = 0.1$) vs CCSDS-122

Mètode	Temps	RAM	Ratio	PSNR	MSE
CCSDS 122 (Lossless)	12.6 s	25 MB	1.79:1	∞	0.00
CCSDS 122 (Near-LL)	12.7 s	25 MB	1.79:1	103.5 dB	0.19
SORTENY C	304 s	87 MB	$\sim 2.5:1^*$	76.7 dB	91.3

*Ratio estimat basat en entropia (sense codificador aritmètic)

✓ CCSDS guanya en:

- Velocitat (24x més ràpid)
- Memòria (3x menys)
- Qualitat lossless perfecta

✓ SORTENY guanya en:

- Ratio de compressió potencial
- **Adaptabilitat** (reentrenable)
- Actualitzable en vol

Conclusions

Conclusions Principals

✓ Èxits Aconseguits

1. Viabilitat Demostrada

- RAM: 796 MB → 87 MB (-90%)
- El sistema ja no col·lapsa

2. Qualitat Preservada

- PSNR idèntic (76.73 dB)
- Paritat 99.98%

3. Rendiment Millorat

- Speedup 1.47x
- 4 nuclis al 338%

💡 Aprendentatges Clau

- La **memòria** és la barrera principal, no la CPU
- És possible portar DL a sistemes embarcats **amb enginyeria adequada**
- Les transformades apreses ofereixen **flexibilitat** vs fixes

⚠ Limitacions

- Codificador entròpic pendent
- CCSDS encara és més ràpid

Codificador Entròpic

Integrar Montsec al mateix executable C per mesurar ràtios reals (bpp).

Acceleració HW

Optimització manual amb intrínsecs NEON o Arm Compute Library.

Validació en Òrbita

Portar a CubeSat real: buit tèrmic, radiació (SEUs), TRL elevat.

Gràcies per la vostra atenció!

Preguntes?

 1635157@uab.cat

Slides Addicionals

Arquitectura Detallada SORTENY

Etapes del Compressor:

1. (A) Transformada Espectral:

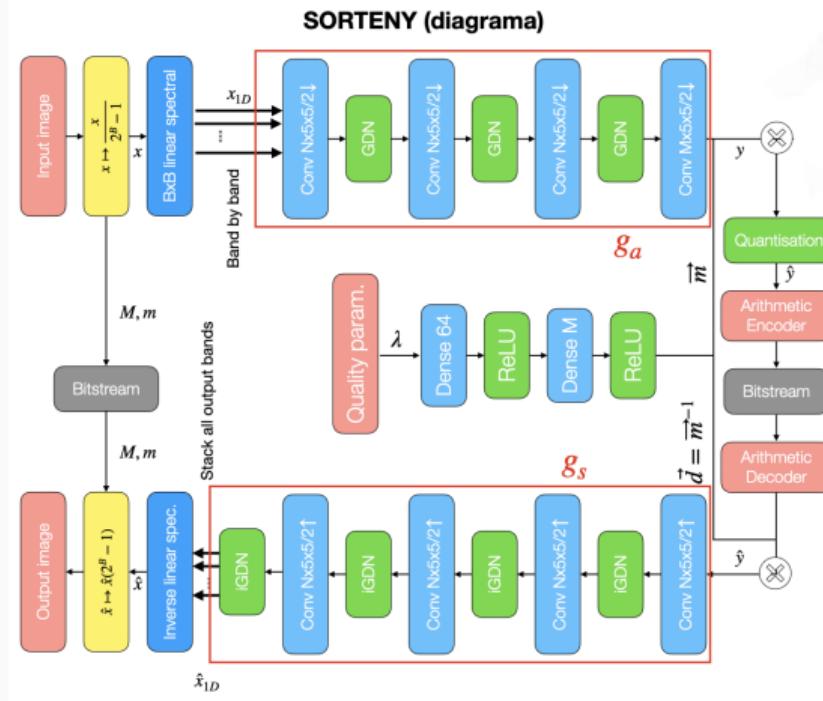
- $x' = Ax + b$
- Similar a PCA/KLT
- Descorrelació de bandes

2. (B) CNN Espacial:

- Conv2D amb stride
- Normalització GDN
- Downsampling progressiu

3. (C) Modulació:

- $\hat{Y} = \text{round}(Y \cdot M(\lambda))$
- Control de qualitat/taxa



Detall: Normalització GDN

Fórmula Matemàtica:

$$y_i = \frac{x_i}{\sqrt{\beta_i + \sum_j \gamma_{ij} x_j^2}}$$

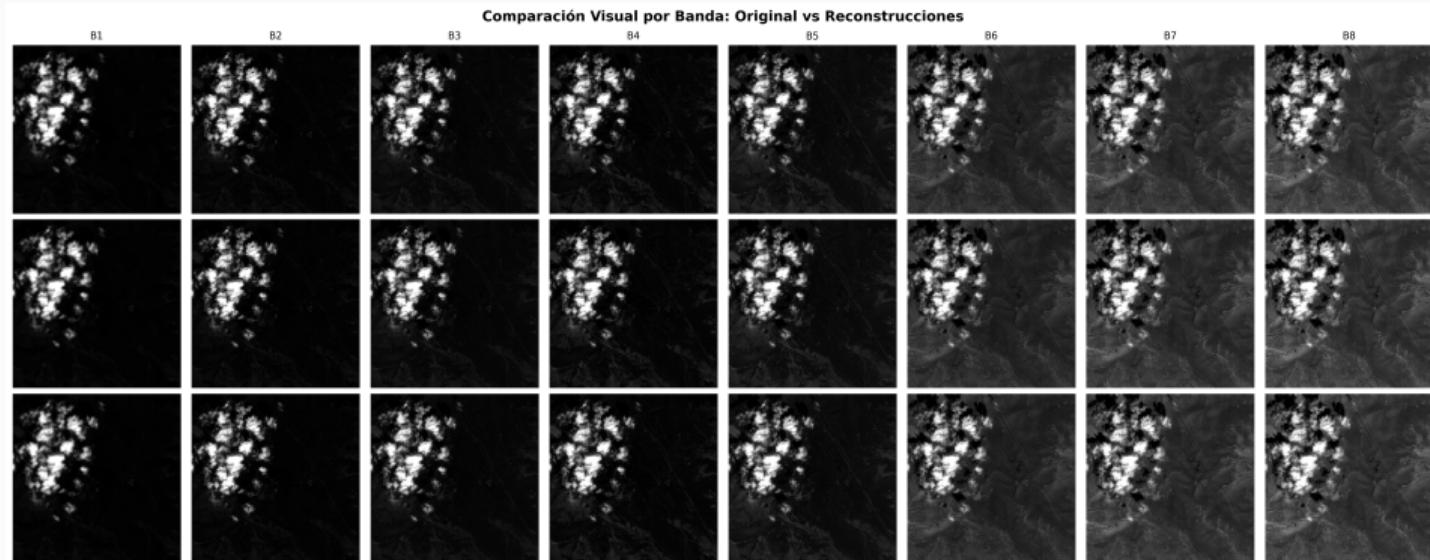
Implementació en C:

- Paràmetres β_i, γ_{ij} apresos
- Gestió de ϵ per estabilitat
- Bucles paral·lelitzats amb OpenMP

Per què GDN i no ReLU?

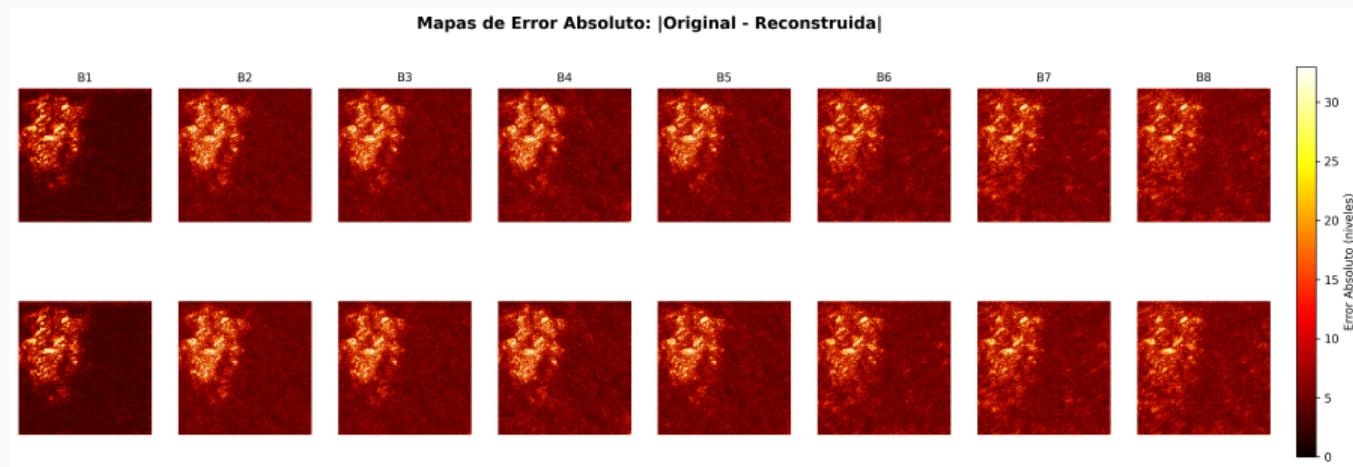
- “Gaussianitza” les dades
- Coeficients més independents
- Millor per a codificació entròpica

Comparativa Visual Completa



Dalt: Original — Mig: SORTENY C — Baix: SORTENY Python

Mapes d'Error



Error absolut —Original - Reconstruïda—. Distribució uniforme, sense artefactes.

Raspberry Pi 3B+ com a Simulador de CubeSat

Per què Raspberry Pi?

- ARM Cortex-A53 = mateixa arquitectura que Xilinx Zynq UltraScale+ (OBCs reals)
- 1 GB RAM simula restriccions reals
- microSD lenta → simula flash de vol
- Cost baix per experimentació

Validació:

- NASA té guies per Raspberry Pi a l'espai
- Estudis IEEE confirmen Cortex-A53 per processament d'imatges a bord
- Codi portable a HW de vol real