Implementació d'un compressor basat en xarxes neuronals per a ús en satèl·lit

Informe de Progrés I

Cristhian Omar Añez López
1635157

Novembre 2025

Contents

1 Objectius

Aquesta secció recull els objectius definits en l'informe inicial.

1.1 Objectiu General

Desenvolupar i validar una versió eficient del compressor SORTENY orientada a plataformes amb recursos limitats, minimitzant l'impacte de les restriccions de càlcul i memòria en el rendiment i la qualitat de la compressió.

1.2 Objectius Específics

a) **Implementació i disseny.** Implementar la inferència del compressor SORTENY en llenguatge C, amb una arquitectura clara i portable, adaptada a l'ordinador de placa utilitzat com a plataforma de simulació.

b) **Identificació i optimització de colls d'ampolla.** Localitzar de manera sistemàtica els colls d'ampolla de càlcul i memòria (transformacions, modulació/codificació, E/S) i aplicar-hi optimitzacions orientades a l'eficiència.

c) **Comparativa de rendiment i qualitat.** Mesurar les prestacions del sistema (ràtio de compressió, PSNR, latència i consum) i comparar-les amb solucions representatives basades en els estàndards CCSDS, mantenint el mateix entorn experimental.

d) **Proposta de millores.** Formular i validar millores de baix cost computacional que redueixin la càrrega de càlcul preservant la qualitat percebuda.

2 Planificació Prevista

La planificació original del TFG es divideix en cinc fases principals:

1. **Revisió dels softwares de compressió i de les capacitats hardware:** Estudi dels estàndards CCSDS i SORTENY, i anàlisi de les restriccions de la plataforma de simulació (Raspberry Pi).
2. **Implementació en C:** Traslladar la lògica de SORTENY a C per obtenir una primera versió funcional que compili i produeixi sortides coherents.
3. **Preparació de l'entorn de treball:** Posada a punt de la Raspberry Pi amb les eines necessàries per executar i provar la versió en C.
4. **Avaluació experimental:** Mesura de resultats (ràtio, qualitat, temps) i comparativa amb la referència (CCSDS 122).
5. **Documentació:** Elaboració de l'informe final i conclusions.

3 Realització dels Objectius (Fases 1 i 2)

Aquesta secció cobreix la feina realitzada durant les dues primeres fases de la planificació.

3.1 Fase 1: Revisió de Tecnologies i Estat de l'Art

L'objectiu d'aquesta fase és realitzar un estudi de l'estat de l'art, analitzant tant la solució proposada (SORTENY) com els estàndards actuals (CCSDS), i definir les restriccions de maquinari de la plataforma de simulació.

3.1.1 Anàlisi del Compressor SORTENY

SORTENY (*Spectral Orthogonal Transform Encoder*) és un algorisme de compressió d'imatges amb pèrdua (*lossy*), dissenyat específicament per a dades d'observació de la Terra, com ara imatges multiespectrals i hiperespectrals. A diferència dels compressors tradicionals (com JPEG o CCSDS) que utilitzen transformades matemàtiques fixes (ex. DCT, Wavelet), SORTENY es basa en el paradigma de compressió apresa (*learned compression*). Utilitza una arquitectura de xarxa neuronal de tipus *autoencoder*, entrenada sobre un gran conjunt de dades de satèl·lit (com Sentinel-2) per optimitzar simultàniament la transformació de la imatge i la seva codificació d'entropia. El procés de compressió és un *pipeline* de diverses etapes dissenyat per explotar eficientment les redundàncies tant espectrals com espacials.

3.1.1.1 Etapa 1: Transformada Espectral (Descorrelació de Bandes)

El primer desafiament en dades hiperespectrals és l'alta redundància entre les bandes; una mateixa escena a 700nm és molt similar a la de 705nm.

SORTENY aborda això aplicant una "Transformada Espectral" apresa. Aquesta etapa funciona de la següent manera:

- **Procés:** Cada píxel es representa com un vector $x \in \mathbb{R}^B$, on B és el nombre de bandes (ex. 5 o 7). Aquest vector es multiplica per una matriu de transformació $A \in \mathbb{R}^{B \times B}$ que ha estat apresa pel model.
- **Objectiu:** L'operació, $x' = A \cdot x$, és funcionalment equivalent a una capa `Dense` de TensorFlow. El model aprèn una matriu de transformació que descorrelaciona la informació de les bandes. La xarxa de síntesi (el descodificador) aprendrà la matriu inversa A^{-1} per a la reconstrucció.
- **Sortida:** S'obté una nova imatge de $B \times H \times W$, on les noves bandes (o "components") són una combinació lineal de les originals, optimitzades per a la compressió.

3.1.1.2 Etapa 2: Transformada Espacial (Anàlisi i Compressió)

Un cop descorrelacionades les bandes, l'arquitectura aplica la compressió espacial.

- **Procés:** El tensor de $B \times H \times W$ es reordena i es tracta com un *batch* de B imatges d'un sol canal ($B \times (1 \times H \times W)$). Aquest *batch* s'introdueix en una única Xarxa Neuronal Convolucional (CNN) anomenada `AnalysisTransform`.
- **Arquitectura:** Aquesta xarxa consisteix en múltiples capes de **convolució 2D** (ex. amb kernels 5x5) i ***strides*** (passos) de 2.
- **Efecte:** L'ús de *strides* redueix progressivament les dimensions espacials (ex. de 512×512 a 256×256, etc.), comprimint la informació espacial en un conjunt més dens de mapes de característiques.
- **Activació GDN:** Entre les capes, s'utilitza una **GDN** (*Generalized Divisive Normalization*). Aquesta funció normalitza la resposta de cada neurona x_i dividint-la per l'activitat de les seves veïnes, modelant millor l'estadística de les imatges. La seva forma general és:

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

  On β i γ són paràmetres entrenables que estabilitzen la normalització. Això és clau per a la compressió, ja que ajusta el senyal per a una quantització més eficient.

3.1.1.3 Etapa 3: Modulació i Quantització (Control de Qualitat)

El resultat de l'Etapa 2 és un tensor latent `Y`. Perquè la compressió sigui *lossy*, aquest tensor de punt flotant ha de ser quantitzat (arrodonit a enters).

- **Procés:** Una petita xarxa anomenada `ModulatingTransform` pren el paràmetre de qualitat desitjat (lambda, λ) i genera un factor d'escala M.

- **Efecte:** Aquesta etapa aplica una **quantització escalar** multiplicativa. El tensor latent `Y` es multiplica per aquest factor abans de ser arrodonit:

$$\hat{Y} = \text{round}(Y \cdot M(\lambda))$$

  Un lambda λ alt resulta en un factor M major, el que porta a una quantització més fina (major qualitat, més bits), mentre que un λ baix resulta en una quantització més agressiva (menor qualitat, menys bits).

- **Sortida:** Un tensor d'enters (símbols) `Y_hat`.

3.1.1.4 Etapa 4: Codificació d'Entropia (Generació del Bitstream)

L'etapa final converteix el tensor d'enters `Y_hat` en un fitxer binari. SORTENY utilitza una tècnica avançada anomenada *hiper-prior* per estimar les probabilitats dels símbols.

- **Autoencoder d'Hiper-Prior:** En paral·lel, una segona xarxa neuronal (la `HyperAnalysisTransform`) analitza el tensor latent `Y` i genera una representació comprimida de la seva pròpia estructura estadística, `Z` (l'hiper-prior).
- **Model de Probabilitat:** L'objectiu és estimar la probabilitat de cada símbol $P(\hat{Y})$. En lloc d'usar una probabilitat fixa (com en Huffman), el model assumeix que $P(\hat{Y})$ segueix una distribució (ex. Gaussiana) els paràmetres de la qual (μ, σ) són, al seu torn, generats per la xarxa a partir de l'hiper-prior `Z`.
- **Codificador Aritmètic de Context:** El *bitstream* es genera mitjançant un "Codificador Aritmètic". Aquest codificador és adaptatiu al context:
  1. El tensor `Z` es quantitza ($\hat{Z}$) i es codifica primer (formant el *bitstream* secundari 'coded_side.bin').
  2. El descodificador usarà $\hat{Z}$ per generar els paràmetres (μ, σ) de la distribució.
  3. El codificador aritmètic usa aquests paràmetres (el "context") per codificar eficientment els símbols $\hat{Y}$ en el *bitstream* principal ('coded_latent.bin').
- **Sortida Final:** Es generen els dos *bitstreams* que, plegats, formen la imatge comprimida.

3.1.1.5 Condicions de Funcionament i Avantatges

SORTENY funciona òptimament en el tipus de dades per al qual va ser entrenat (imatges multiespectrals). El seu avantatge teòric és que, en ser "après", el model complet (la transformada espectral, les convolucions espacials i el model de probabilitat) està optimitzat d'extrem a extrem (*end-to-end*) per a la màxima eficiència de compressió, superant potencialment els mètodes tradicionals que separen aquests passos.

3.1.2 Anàlisi dels Estàndards CCSDS

El *Consultative Committee for Space Data Systems* (CCSDS) defineix un conjunt d'estàndards per a la compressió de dades a bord de satèl·lits. Per a aquest treball, és crucial identificar la línia base (*benchmark*) més adequada per comparar amb SORTENY.

3.1.2.1 Context: Estàndards Predictius (CCSDS 121 i 123)

L'estàndard CCSDS defineix algorismes predictius dissenyats per a una baixa complexitat computacional, ideals per a maquinari de vol (FPGAs) [2, 13].

- **CCSDS 121.0-B-2 (Compressió Sense Pèrdua):** És un algorisme *lossless* [13, 15] que utilitza un predictor 2D i un **codificador Rice** adaptatiu [1, 14, 13]. Sovint s'usa com a etapa final de codificació d'entropia per a altres estàndards [12].
- **CCSDS 123.0-B-2 (Compressió Hiperespectral):** Aquest és l'estàndard *lossless* i *near-lossless* dissenyat específicament per a dades hiperespectrals [2, 3, 4]. És un algorisme predictiu *single pass* [6] que estima cada píxel basant-se en un veïnatge 3D (espacial i espectral) i després codifica només el residu de l'error [2], sovint usant el codificador Rice de CCSDS 121 [12].

Si bé CCSDS 123 és l'estàndard per a dades hiperespectrals, el seu enfocament predictiu i la seva optimització per a compressió *lossless* el converteixen en una comparativa menys directa per a SORTENY, que és un compressor *lossy* basat en transformades.

3.1.2.2 Benchmark Principal: CCSDS 122.0-B-1 (Compressió amb Pèrdua)

Seguint les condicions establertes al treball, s'estableix l'estàndard CCSDS 122 com el *benchmark* principal, ja que permet una comparació més adequada de les filosofies de disseny. CCSDS 122 està dissenyat per a la compressió d'imatges 2D (monobanda) i ofereix modes amb pèrdua (*lossy*) i sense pèrdua (*lossless*) [11, 8]. La seva arquitectura és conceptualment similar a JPEG 2000 [8] i consta de dues etapes:

1. **Etapa 1: Transformada Discreta de Wavelet (DWT)**: Utilitza filtres matemàtics fixos (ex. 9/7 biorthogonal) per descompondre la imatge en sub-bandes de freqüència, descorrelacionant eficaçment la informació espacial [8, 10].
2. **Etapa 2: Quantització i Codificació (BPE)**: Els coeficients de la DWT es quantitzen (el pas on s'introdueix la pèrdua o *loss*) i després es codifiquen bit a bit mitjançant un "Codificador de Pla de Bits" (BPE) [10, 9].

3.1.2.3 Comparativa de Condicions: CCSDS 122 vs. SORTENY

La comparativa clau d'aquest treball es centrarà en aquestes dues arquitectures, ja que ambdues resolen la compressió amb pèrdua mitjançant transformades, però amb filosofies oposades:

- **CCSDS 122** utilitza una **transformada matemàtica fixa (DWT)**. La transformada és coneguda, ràpida i eficient, però no està optimitzada per al tipus de dades específic. La compressió s'aconsegueix separant la transformada de la quantització i la codificació d'entropia.
- **SORTENY** utilitza una **transformada "apresa" (Autoencoder)**. El model complet (la transformada espectral, les convolucions espacials i el model de probabilitat) està optimitzat d'extrem a extrem (*end-to-end*) per a la màxima eficiència de compressió en un tipus de dades específic (imatges de satèl·lit).

L'objectiu serà comparar l'eficiència de la solució "apresa" i d'alta complexitat (SORTENY) enfront de la solució "fixa" i de baixa complexitat (CCSDS 122) en un entorn de maquinari limitat.

3.1.3 Anàlisi del Maquinari (Raspberry Pi)

L'objectiu és implementar una versió de SORTENY optimitzada per a maquinari amb recursos limitats, similar al que es trobaria en un CubeSat. Per simular aquest entorn, s'utilitzarà un ordinador monoplaca (SBC) tipus Raspberry Pi. És crucial entendre les limitacions d'aquesta plataforma per fixar expectatives realistes. Prenem com a referència les especificacions de la "Raspberry Pi 4 Model B", un model representatiu i àmpliament disponible, segons la seva documentació oficial [16, 17]:

- **Processador (CPU):** Utilitza un SoC (System on a Chip) Broadcom BCM2711, que inclou una CPU ARM Cortex-A72 (ARM v8) de 64 bits i quatre nuclis (quad-core) a 1.5GHz [16, 18]. Aquesta arquitectura ARM és fonamentalment diferent dels processadors x86 dels ordinadors d'escriptori i portàtils.
- **Memòria (RAM):** La placa està disponible en configuracions d'1GB, 2GB, 4GB o 8GB de RAM LPDDR4 [16, 18]. Aquesta quantitat de memòria és una restricció severa. Processar imatges hiperespectrals completes, que poden ocupar diversos gigabytes, és inviable. Per tant, qualsevol implementació ha de processar la imatge per blocs o *patches*.
- **Unitat Gràfica (GPU):** Integra una GPU Broadcom VideoCore VI [17, 16]. Aquesta GPU està dissenyada principalment per a multimèdia (suporta descodificació H.265 4Kp60 [16]) i gràfics (OpenGL ES 3.0 [16]), però no té les capacitats de còmput de propòsit general (GPGPU) que es troben a les GPUs de NVIDIA (CUDA).
- **Emmagatzematge:** El sistema operatiu i les dades es carreguen des d'una targeta microSD [16, 18]. La velocitat de lectura/escriptura d'aquesta targeta és un coll d'ampolla significatiu comparada amb els discos SSD d'un PC.

3.1.3.1 Implicacions

L'anàlisi del maquinari defineix les nostres restriccions de disseny i els objectius principals de la implementació:

1. **Eliminació de Dependències Pesades:** La implementació base de SORTENY utilitza l'API de C de TensorFlow per executar la inferència del model. Aquesta llibreria és de grans dimensions, està dissenyada per a arquitectures x86 i GPUs NVIDIA, i no és viable per a un sistema embarcat amb recursos limitats. La implementació en C ha de ser nativa, pura i sense dependències externes pesades.
2. **Gestió de Memòria:** La limitada RAM de la plataforma (ex. 4GB o 8GB) [16] prohibeix carregar imatges hiperespectrals completes, que poden ocupar diversos gigabytes. La solució ha d'adoptar un enfocament de *streaming* o basat en blocs (*patch-based*), processant la imatge en petits blocs que sí que càpiguen a la memòria disponible.
3. **Optimització d'E/S:** El *pipeline* de compressió de la implementació base està dividit en dos programes executables separats: una primera etapa que realitza la transformació de la xarxa neuronal i escriu els seus resultats (tensors intermedis) a l'emmagatzematge; i una segona etapa que ha de tornar a llegir aquests fitxers del disc per realitzar la codificació d'entropia. Donat el coll d'ampolla que suposa l'emmagatzematge en una targeta microSD [18], aquesta estratègia d'E/S és un punt crític. Una implementació eficient ha de fusionar aquests passos i mantenir les dades en memòria, passant la sortida de la xarxa neuronal directament al codificador aritmètic.

3.1.4 Conclusions de la Fase 1 i Expectatives

L'anàlisi de les tecnologies de programari i les plataformes de maquinari defineix el punt de partida i uns objectius a executar. La comparativa entre SORTENY i els estàndards CCSDS revela la divisió d'aquest treball:

- **Estàndard Tradicional (CCSDS 122):** Ofereix una solució *lossy* de complexitat moderada, basada en una transformada matemàtica fixa (DWT) [11, 8]. És un estàndard provat i eficient.
- **Enfocament "Après" (SORTENY):** Ofereix una solució *lossy* d'alta eficiència que promet ràtios de compressió superiors. Això s'aconsegueix mitjançant una arquitectura d'*autoencoder* optimitzada *end-to-end* (transformades espectrals, convolucionals i codificació d'entropia adaptativa).

No obstant això, la implementació de referència de SORTENY proporcionada per a aquest treball no és adequada per a un entorn embarcat. L'anàlisi del maquinari de la Raspberry Pi 4, la nostra plataforma de simulació, ha revelat les restriccions crítiques que defineixen els objectius d'aquest projecte [16, 17]:

1. **Dependència de Programari Pesat:** La implementació actual de SORTENY en C depèn de `libtensorflow` per a la inferència, una llibreria de grans dimensions i incompatible amb els requisits d'un sistema embarcat sense una GPU de còmput.
2. **Limitacions de Memòria (RAM):** La RAM disponible (ex. 4GB o 8GB) [16] és insuficient per a carregar imatges hiperespectrals completes. Per tant, el disseny de programari ha de processar les dades per blocs (*patch-based*).
3. **Coll d'Ampolla d'E/S (Entrada/Sortida):** El *pipeline* de C actual escriu tensors intermedis a l'emmagatzematge (targeta microSD) [18], per a després ser llegits per un segon programa. Aquesta operació d'E/S en un emmagatzematge lent és un coll d'ampolla crític.

La conclusió d'aquesta fase és que existeix una clara oportunitat de millora. L'objectiu serà dissenyar i implementar una nova versió de SORTENY en C que:

- **Sigui autònoma:** Reemplaci la dependència de `libtensorflow` per una reimplementació nativa en C de les operacions de la xarxa (convolucions, GDN, etc.).
- **Sigui eficient en memòria:** Operi en mode *streaming* (per blocs), passant les dades de la xarxa neuronal directament al codificador aritmètic.
- **Elimini el coll d'ampolla d'E/S:** Fusioni les etapes de transformació i codificació en un únic programa, evitant l'escriptura i lectura de fitxers intermedis al disc.

3.2 Fase 2: Anàlisi i Implementació Nativa en C

Un cop identificades les restriccions de maquinari i els colls d'ampolla de la implementació de referència a la Fase 1 (dependència de `libtensorflow`, ús de fitxers intermedis d'E/S), la Fase 2 s'ha centrat en l'Objectiu Específic (a): "Implementació i disseny".

L'objectiu principal d'aquesta fase ha estat crear una implementació nativa en llenguatge C pur que substitueixi el *pipeline* original i aconseguir la paritat numèrica amb el model de referència de Python/TensorFlow.

Per aconseguir-ho, s'ha desenvolupat un nou projecte amb una arquitectura de validació robusta, separant l'extracció de pesos (Python) de la implementació de la inferència (C).

3.2.1 Metodologia: Extracció de Pesos

Per trencar la dependència amb `libtensorflow`, el primer pas ha estat extreure els pesos entrenats del model.

- S'ha creat un *script* (`src/python/pesos.py`) que carrega el `SavedModel` original (`models/SORTENY_Sentinel2_model`) utilitzant TensorFlow.
- Aquest *script* itera per totes les capes de la xarxa neuronal (transformada espectral, convolucions, paràmetres GDN i capes denses de modulació).
- Tots els pesos (kernels, bias, beta, gamma) s'exporten com a fitxers binaris plans (`float32`) a la carpeta `weights/pesos_bin/`.
- Finalment, es genera un índex (`weights_index.tsv`) que mapeja cada nom de pes al seu fitxer `.bin`, *dtype* i dimensions, per tal que el codi C pugui carregar-los.

3.2.2 Implementació de la Inferència Nativa en C

El nucli d'aquesta fase ha estat la reimplementació de totes les operacions de la xarxa neuronal en C pur, sense dependències externes (excepte `libm`).

- **Càrrega de Pesos (`sorteny_model.c`):** S'ha implementat un carregador que llegeix l'índex `.tsv`, reserva la memòria necessària (`malloc`) per a l'estructura `SORTENY_Model`, i carrega tots els pesos binaris a la RAM.
- **Implementació de Capes (`sorteny_layers.c`):** S'han recreat les funcions matemàtiques de les capes de TensorFlow:
  - **apply_dense:** Multiplicació matriu-vector per a la transformada espectral i la moduladora.
  - **apply_conv2d:** Convolució 2D que replica la semàntica de correlació i el *padding* asimètric SAME de TensorFlow.
  - **apply_gdn:** Implementació de la fórmula exacta de la *Generalized Divisive Normalization* (GDN), incloent la gestió d'*epsilon* i l'orientació dels pesos *gamma*.
  - **apply_relu:** Activació estàndard $\max(0, x)$.
- **Pipeline Principal (`main.c`):** S'ha creat un nou programa orquestrador que substitueix el `transform.c` original. Aquest programa:
  1. Carrega el model (`load_model_weights`) i la imatge (`load_image_bsq_u16_to_planar_f32_ex`).
  2. Reserva memòria per a tots els tensors intermedis.
  3. Executa el *pipeline* de 5 etapes en C: Transformada Espectral → Normalització → Analysis Transform (bucle de 8 bandes) → Modulating Transform → Quantització (amb redondeig).
  4. Desa la sortida final (`Y_hat_c.bin`).

3.2.2.1 Metodologia de Validació per Etapes

Per garantir la paritat numèrica, s'ha implementat un *pipeline* de validació:

- **Generació de la "Veritat Absoluta" (`validar_python.py`):** Aquest *script* de Python carrega el model de TF, però en lloc d'executar-lo d'un sol cop, replica exactament el *pipeline* del C (capa per capa, incloent la normalització manual i la reordenació de tensors). La seva sortida (`python_ground_truth.bin`) és el resultat teòricament perfecte.

- **Sistema de Depuració (`DUMP`):** Tant el `main.c` com el `validar_python.py` responen a variables d'entorn (ex. `DUMP_STAGES=1`). Això permet desar els tensors intermedis de cada etapa (ex. `gdn0_c.bin` vs. `gdn0_py.bin`).

- **Comparació Numèrica (`compare_products.py`):** Aquest *script* compara els binaris de C i Python, etapa per etapa, i informa de les diferències numèriques.

3.2.2.2 Resultats de la Fase 2

L'execució del *pipeline* de validació ha estat un èxit i ha permès identificar i corregir errors fins a assolir la paritat numèrica (considerant les diferències de coma flotant):

- Les diferències en els tensors de coma flotant previs al redondeig són mínimes (ex. `Y_float_c vs Y_float_py` max $\approx 3.8 \times 10^{-3}$).

- S'ha implementat un redondeig bancari (*half-to-even*) en C (`USE_HALF_EVEN=1`) per replicar el comportament de `tf.round`.

- Amb aquest redondeig, la sortida final (`Y_hat`) és gairebé idèntica, amb diferències màximes de ± 1 només en casos frontera de redondeig.

Aquesta fase conclou amb una implementació nativa en C validada i funcional, que elimina la dependència de `libtensorflow` i resol els principals colls d'ampolla de disseny.

4 Conclusions i Pròxims Passos

4.1 Conclusions del Període

Aquest primer informe de progrés ha cobert la realització de les Fases 1 i 2 de la planificació, centrades en l'anàlisi de viabilitat i la implementació d'un prototip funcional.

- **Fase 1 (Revisió):** L'anàlisi de l'estat de l'art ha confirmat la idoneïtat de la comparativa entre l'enfocament d'alta complexitat "après" de SORTENY i l'enfocament de baixa complexitat "fix" de l'estàndard CCSDS 122 (basat en DWT). L'anàlisi del maquinari (Raspberry Pi) va identificar tres colls d'ampolla crítics a la implementació de referència: la dependència de `libtensorflow`, les limitacions de gestió de memòria (RAM) i, el més important, el coll d'ampolla d'E/S (I/O) causat per l'escriptura de fitxers intermedis a la targeta microSD.

- **Fase 2 (Implementació):** S'ha completat amb èxit l'objectiu de dissenyar una nova implementació nativa en C (`sorteny_compressor`). Aquesta implementació replica matemàticament tot el *pipeline* de la xarxa neuronal (Transformada Espectral, Convolucions, GDN i Modulació) sense cap dependència externa. S'ha creat un entorn de validació robust que demostra la paritat numèrica (amb la precisió de coma flotant) entre la sortida del codi C i la referència de Python.

Com a conclusió, s'ha resolt el coll d'ampolla de la dependència de programari (`libtensorflow`) i s'ha validat una base de codi C nativa i eficient.

4.2 Pròxims Passos (Fases 3 i 4)

El treball realitzat a la Fase 2 ha generat el tensor latent final (`Y_hat`), però encara no el comprimeix en un *bitstream*. El *pipeline* actual encara depèn d'escriure aquesta sortida al disc, la qual cosa no resol el coll d'ampolla d'E/S.

Els pròxims passos es centraran a resoldre aquest problema i a realitzar l'avaluació experimental:

1. **Fase 3: Integració i Optimització del Compressor**
   - **Fusió de Components:** El pròxim objectiu és **fusionar** la implementació nativa de la xarxa neuronal (`main.c` / `sorteny_layers.c`) amb el codificador aritmètic (`encoder.c` del projecte original). L'objectiu és crear un únic executable que realitzi la transformació i la codificació d'entropia en memòria, passant les dades directament del tensor `Y_hat` al codificador aritmètic sense escriure mai fitxers intermedis al disc.
   - **Optimització de Memòria:** Paral·lelament a la fusió, s'adaptarà el codi perquè operi en mode *patch-based* (processament per blocs), en lloc de carregar la imatge sencera a la RAM, resolent així la limitació final de memòria.
2. **Fase 4: Avaluació Experimental i Comparativa**
   - **Desplegament:** Un cop es disposi de l'executable fusionat i optimitzat, es desplegarà a la plataforma de simulació (Raspberry Pi).
   - **Mesures de Rendiment:** S'executaran les proves per mesurar les mètriques clau: latència (temps de compressió), ús de memòria (RAM), ràtio de compressió i qualitat (PSNR).
   - **Comparativa amb CCSDS 122:** Finalment, aquests resultats es compararan directament amb el *benchmark* establert (CCSDS 122) per validar la hipòtesi del TFG.

References

[1] Santos, L., Gomez, A., & Sarmiento, R. "Implementation of CCSDS Standards for Lossless Multispectral and Hyperspectral Satellite Image Compression," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 3, pp. 2120-2133, 2020.

[2] Johansen, T. A., Hjelmstad, E. L. M. S. E. L., & Lande, T. S. "An Efficient Real-Time FPGA Implementation of the CCSDS-123 Compression Standard for Hyperspectral Images," En *2018 26th European Signal Processing Conference (EUSIPCO)*, Rome, Italy, 2018, pp. 1197-1201.

[3] Báscones, D., González, C., & Mozos, D. "Parallel Implementation of the CCSDS 1.2.3 Standard for Hyperspectral Lossless Compression," *Remote Sensing*, vol. 9, no. 10, p. 973, 2017.

[4] Rodriguez-Moreno, M. A., Galiano-Ortega, D., Chaves-Cortes, R. C., & Toledo, F. J. "Hardware Implementation of the CCSDS 1.2.3-B-2 Near-Lossless Compression Standard Following an HLS Design Methodology," *Electronics*, vol. 10, no. 16, p. 1959, 2021.

[5] Chaves-Cortes, R. C., Galiano-Ortega, D., Rodriguez-Moreno, M. A., & Toledo, F. J. "Scalable Hardware-Based On-Board Processing for Run-Time Adaptive Lossless Hyperspectral Compression," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 10, pp. 7855-7867, 2019.

[6] Gómez-de-Gabriel, J. M., Guerra-T., M., Gómez-A., A., López, R., & López, S. "A Smart Compression Approach Based on the CCSDS 123.0-B-2 Standard for CHIME," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2022.

[7] Zeng, X., Huang, H., Lv, D., Zeng, M., & Ping, Y. "Optimization of Compression Algorithm for Snapshot Mosaic Hyperspectral Images Based on CCSDS 123 Standard," En *Proc. SPIE 12747, International Conference on Optical and Photonic Engineering (icOPEN 2023)*, 2023, Art. no. 127471G.

[8] Alma Technologies, "CCSDS 122.0-B-1 Encoder IP Core - Product Brief," [Online]. Available: https://www.alma-technologies.com/ip-core.CCSDS-122-E.

[9] Agudo-G., J. I., Serra-Cano, J. F., & G.-Diaz, E. "Discrete wavelet transform fully adaptive prediction error coder: Image data compression based on CCSDS 122.0 and fully adaptive prediction error coder," *IEEE Latin America Transactions*, vol. 13, no. 10, pp. 3254-3260, 2015.

[10] Zicter, P., Schiavon, G. G. G., Fanucci, L., & Fossi, L. "A 13.3 Gbps 9/7M Discrete Wavelet Transform for CCSDS 122.0-B-1 Image Data Compression on a Space-Grade SRAM FPGA," *Electronics*, vol. 9, no. 8, p. 1234, 2020.

[11] Consultative Committee for Space Data Systems (CCSDS). "Image Data Compression." *CCSDS 122.0-B-2. Blue Book. Issue 2.* Washington, D.C.: CCSDS, 2017.

[12] Cobham Gaisler, "CCSDS 121/123 Lossless Compression - Product Brief," [Online]. Available: https://www.gaisler.com/products/ccsds-121-123-lossless-compression.

[13] Tsigkanos, A., G., D. G., Paliouras, V., & Goutis, C. "A Reconfigurable FPGA Implementation of CCSDS 121.0-B-2 Lossless Data Compression," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, no. 4, pp. 1949-1961, 2019.

[14] Cabral, L., Matos, P., Evans, G., & Santos, J. "EFFICIENT DATA COMPRESSION FOR SPACECRAFT INCLUDING PLANETARY PROBES," En *Proc. 7th International Planetary Probe Workshop*, Barcelona, Spain, 2010.

[15] Consultative Committee for Space Data Systems (CCSDS). "Lossless Data Compression." *CCSDS 121.0-B-3. Blue Book. Issue 3.* Washington, D.C.: CCSDS, 2020.

[16] Raspberry Pi Ltd., "Raspberry Pi 4 Model B Product Brief," 2025. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf

[17] Raspberry Pi Ltd., "Raspberry Pi 4 Model B Datasheet," 2024. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[18] Raspberry Pi Ltd., "Raspberry Pi 4 Model B - Specifications," [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[19] RS Components, "Raspberry Pi 4 Computer Model B - Technical Datasheet," [Online]. Available: https://docs.rs-online.com/b1fb/0900766b816fa153.pdf.