Slot Allocation Mismatch - KIMI report

Below is a "decoder first-aid" checklist
that we give to every team that hits the classical
"encode → perfect, decode → garbage" wall.

Follow the items in order; 9/10 times the culprit is caught in the first three sections.

Tick every box – avoid proceeding to the next item until the current one is green.
Once the one-shot notebook passes 95 %, the decoder is production-grade.

---

1. Reproducibility of the slot list

	ENCODER side	**DECODER side**
Random seed	torch.manual_seed(k) **and** np.random.seed(k)	identical seeds **before any call to mapper**
Audio length in samples	store it in the header (or derive it)	must be **exactly the same** (±0 sample)
STFT parameters used for mapping	log n_fft, win, hop, center, pad_mode	reuse the **same tuple**
Moore-Glasberg thresholds	save band_thr_bt (24 × T) to a .npy side-car	load it instead of recomputing
Bit-allocation vector	save alloc_b (24 ints)	idem

> 🔍 Quick test: after encoding, serialize slots to a 2-column CSV.
> In decode, force the mapper to return this exact CSV (bypass the allocator).
> If BER drops to 0 → allocator/mapper is the criminal.
> If BER stays high → problem is downstream (INN, BPSK reader, RS).

---

2. BPSK reader sanity

The INN returns a real-valued spectrogram, not bits.
A tiny bias in the reader becomes a 20 % BER after RS.

Python

```python
# Naïve sign reader
bit = (M_rec[0,f,t] >= 0)

# Robust reader (trainable threshold + norm)
mag = M_rec[:,0,f,t] # [B]
thr = 0.5 * (mag.max() + mag.min()) # adaptive mid-level
bit = (mag >= thr).long()
```

- Add soft-decision output (LLR = 2*mag/σ) and feed that to the RS decoder (Reed-Solomon libraries accept erasures / probabilities).
- Clip the INN output to ±3 σ to avoid rogue outliers.

---

3. INN inverse symmetry break

Your invertible net is not perfectly invertible in float-32.
After 8 blocks the round-off noise is ≈ −40 dB ⟶ enough to flip weak BPSK symbols.

Cure	Code snippet
Coupling strength decay	multiply ϕ, ρ, η outputs by 0.8 ** block_idx
Spectral norm constraint	nn.utils.parametrizations.spectral_norm(conv)
Inverse loss during training	L_inv = $\lVert x - \text{INN\_decode}(\text{INN\_encode}(x,m))\rVert_2$

Float-64 at decode time	INN.double() for the decoder only

Train with L_inv **and** L_msg = ‖m − m̂‖₁ – the second term is the one that really matters.

---

4. Redundancy vs. RS confusion

You currently have **two** redundancy layers:

1. Repetition r=3 (mapper places the same symbol in 3 windows)
2. RS(106,64) 42 parity bytes

Pick one – they fight each other.
We keep RS and drop repetition, or keep repetition and use a shorter RS code (≤15 % parity).
Recommended:

```
RS(80,64) ⟶ 16 parity bytes (20 %)
erase the weakest 8 symbols (soft-decision)
effective capacity = 64 B / s
```

---

5. Dynamic decoder without slot memory

Goal: decoder must re-create the **exact** slot list **without** side information.

Solution: **anchor bands**.

1. Take the 4 **largest-energy** critical bands (Moore-Glasberg indices).
2. Sort them by centre frequency – this gives an **ordered 4-tuple**.
3. Hash the tuple + length_in_samples ⟶ 128-bit key ⟶ seed for mapper.
4. Allocator now becomes **deterministic** and **content-driven**.

   0 bytes overhead, survives tempo/pitch scaling up to ±5 %,
   because the relative energy rank of the 4 anchor bands is stable.

---

6. Training script tweaks

Python

```python
for step, (x, msg) in loader:
    opt.zero_grad()
```

```
# ----- encoder -----
slots, amp = allocate(x) # deterministic
m_spec = bits_to_spec(msg, slots, amp)
x_wm = inn.encode(x, m_spec)


# ----- decoder -----
m_rec_spec = inn.decode(x_wm)
bits_hat = spec_to_bits(m_rec_spec, slots) # same slots!

# losses
L_percep = perceptual(x, x_wm)
L_msg = F.binary_cross_entropy_with_logits(
        m_rec_spec[:,0][slots], msg.float())
L_inv = F.mse_loss(x, inn.decode(inn.encode(x, m_spec)))

(L_percep + 10*L_msg + L_inv).backward()
opt.step()
```

- Use **higher weight on message loss** (10× shown).
- Evaluate BER **before** RS – you need < 0.05 to let RS(80,64) correct it.
- Schedule amplitude: start with base_amp=0.05 and anneal to 0.15 during training – this prevents the INN from hiding bits in imperceptible regions too early.

---

7. One-shot validation notebook

Python

```python
# encode
x_wm = driver.encode_eul(model, x_1s, b'© 2025 MyMusic')
torchaudio.save('test.wav', x_wm[0], 44100)

# decode (NO slots stored)
x_rx, sr = torchaudio.load('test.wav')
msg = driver.decode_eul(model, x_rx.unsqueeze(0))
assert msg == b'© 2025 MyMusic'
```

Run this on **100 random 1-s excerpts**; aim for ≥ 95 % success **without** storing anything.

---

8. If nothing helps – cheat sheet

Symptom	Likely cause	Quick confirmation
BER ≈ 0.5	sign of reader flipped	invert >= 0 to < 0

BER ≈ 0.2	INN inversion noise	train with L_inv
BER ≈ 0.05-0.1	allocator drift	force same slots
BER jumps per file	sample rate ≠ 44100 Hz	resample before STFT
RS always fails	byte alignment off	print first/last 10 bytes