BRIEF TÉCNICO · ENGENHARIA DE SOFTWARE E DE TELECOM

TUBE — Telemetry Under Bounded Error

Codec de telemetria com bound de erro determinístico por canal: a reconstrução vive dentro de um tubo de raio ε em volta do sinal original, amostra por amostra — `|erro| ≤ bound`, por construção, verificado no decode.

Repositório: `github.com/andradeandrey/telemetria` · Go (stdlib + 1 dependência) e JS · todos os números deste documento vêm do benchmark reproduzível do repositório (seed fixa; datasets reais públicos).
Documento gerado a partir do README do projeto · julho/2026

---

1. Sumário executivo técnico
2. Modelo de dados: canais, bounds e a API
3. Os formatos v1–v5 + v4 seek e o pipeline
4. Telecom: bytes no fio, GOP como unidade de perda, tubesec
5. Índice por GOP e queries certificadas
6. Benchmarks completos
7. Limites honestos e hipóteses refutadas
8. Integração: contrato Go⇄JS, ROS2, degraus de adoção

1 Sumário executivo técnico

O que é. O TUBE é um codec de compressão *lossy com garantia* para telemetria numérica. Cada canal declara um erro absoluto máximo tolerável (o *bound*, ex.: temperatura ±0,1 °C, junta de robô ±0,001 rad, ECG ±5 µV). O codec garante `|original − reconstruído| ≤ bound` em **toda amostra** — erro máximo determinístico, não métrica média (não é PSNR, não é RMSE). A garantia é verificada no decode e coberta por teste de propriedade; a coluna `maxerr/bound` do benchmark é 1,00 ✓ em todos os casos.

De onde vem o ganho. Ruído de sensor na mantissa do float64 parece aleatório para qualquer compressor lossless (gzip e Gorilla dão 1,1–1,3× em satélite e robótica). A quantização com bound é quem mata esse ruído; deadband, predição e entropy coding fazem o resto. O resultado em 15 casos (11 com dados reais públicos): **7,2× a 189× sobre o raw**, sempre com a garantia por amostra cumprida.

Placar honesto. Contra os concorrentes rodados de verdade, com o mesmo bound absoluto por canal e a garantia deles verificada por decode:

- **SZ3** (referência acadêmica de compressão error-bounded), inclusive *afinado* — best-of de algoritmo por canal e layout 2D canais×tempo: o TUBE v4 vence **15 de 15** (margens de 1,15× a 4,5×… 2,8× após o tuning do satélite).
- **ZFP** (LLNL, fixed-accuracy): 7–50× pior em série temporal 1D ruidosa — não é o habitat dele.
- **Swinging Door Trending** (o algoritmo dos historians industriais, PI System/CygNet): TUBE ganha por **1,8×–10,1×** nos 15 casos; até o formato com seek ganha em todos.
- **ModelarDB-RS** (o vizinho mais próximo em contrato — lossy com bound por amostra): TUBE vence os 15 casos por 2,1×–9,5× — e a medição revelou que o ModelarDB *viola o próprio bound* em canais f64 de valores grandes (é f32 interno; erro de até $2{,}55\times10^{8}$ vezes o bound em timestamps).
- **BUFF** (PVLDB 2021): TUBE vence os 15 por 1,55×–255× — o preço da grade decimal e da ausência de predição temporal, o trade-off de ser consultável in-situ.

> **A frase defensável — exatamente como está no README:** "melhor taxa com garantia por amostra contra tudo que é publicado e instalável que conseguimos rodar — SZ3 (afinado, 2D), ZFP, swinging door, ModelarDB e BUFF, sempre com o mesmo bound e a garantia verificada dos dois lados". **Não** é "o melhor do mercado": os eixos onde o TUBE não compete são velocidade de encode (assimetria de projeto, como codec de vídeo), maturidade e campos multidimensionais.

Além da taxa, três propriedades que os concorrentes batch não dão neste modo: streaming incremental, **seek por GOP** (decodificar um trecho do meio sem ler o resto) e **perda de pacote limitada ao bloco**. E, sobre o formato com seek: um índice sidecar de 2–4 B/bloco responde "quando o canal passou de X?" tocando só os blocos relevantes, com resposta *certificada* (seção 5), e um envelope criptográfico pós-quântico sela cada GOP autenticando o próprio bound (seção 4).

2 Modelo de dados: canais, bounds e a API

A unidade do TUBE é o **canal escalar**: uma série de `float64` com um bound próprio. Toda estrutura de telemetria vira colunas – o mesmo movimento do colunar analítico, aqui com um contrato de erro por coluna:

```
type Channel struct {
    Name   string    // identificador ("temp", "gyro_x", "p0_yaw")
    Bound  float64   // erro absoluto máximo tolerado (o ε do tubo)
    Data   []float64 // as amostras
    Period float64   // 0 = linear; != 0 = canal circular (ex.: 360 para yaw em graus)
}
```

Mapa campo → canal → bound

TIPO DE CAMPO	VIRA	BOUND TÍPICO (MEDIDO NOS CASOS)
Grandeza contínua (temperatura, pressão, posição, tensão)	1 canal por campo	tolerância de engenharia: temp ±0,1 °C, pressão ±5 Pa, EKF ±2 cm, junta ±0,001 rad
Inteiro que precisa ser exato (contadores, PWM, nº de episódio, fila)	1 canal com bound ±0,5	±0,5 ⇒ o arredondamento devolve o inteiro exato – "lossless de graça" dentro do mesmo pipeline
Grandeza circular (yaw, heading, fase)	1 canal com `Period`	yaw com `Period: 360` : predição, residual e reconstrução operam módulo 360 e a garantia vale na *distância angular* (179,95° e −179,95° distam 0,1°, não 359,9°)
Vetores / arrays / mensagens (IMU xyz, quaternion, N jogadores × pose)	1 canal por componente	ex.: CS2 = 10 jogadores × (x, y, z, yaw, pitch) = 50 canais; drone PX4 = 24 canais de 4 tópicos
Timestamps (quando a cadência real tem jitter)	canal próprio	±0,5 µs nos casos ROS2 e manipulador; no CAN idem – o tempo é dado como outro qualquer
Strings, enums textuais, blobs	**fora do escopo**	payload não numérico segue no container original (ex.: no MCAP ficam os tópicos não numéricos)

O bound é decisão de engenharia de quem conhece o sensor, não do codec. Regra prática usada nos casos: bound abaixo do ruído do sensor ⇒ a perda fica abaixo do que o sensor já mente (ECG real: ±5 µV = ±1 LSB do ADC; mocap: abaixo do ruído do óptico; CAN: ½ passo nativo do fabricante = lossless efetivo).

Semântica da garantia. A quantização usa passo `step = 2×bound` e `round(v/step)` – o bound vale por construção, e o decode ainda mede o erro real contra o original (a coluna de honestidade `maxerr/bound`). Para canais circulares a métrica é a distância no círculo; para `Period == 0` o caminho de código é byte-idêntico ao formato anterior (os vetores do contrato JS não mudaram).

3 Os formatos: v1 → v5 + v4 seek

O pipeline em cinco estágios

1. **Quantização com garantia** – passos de `2×bound` ; `|erro| ≤ bound` por construção.
2. **Deadband** – só transmite quando o valor sai da banda do último valor reconstruído. Sensor parado ≈ zero bytes de payload (satélite: 91,8% das amostras suprimidas).
3. **Predição** – v2/v3: *hold* ou *linear* por bloco (o encoder codifica nos dois modos e fica com o menor). v4/v4s/v5: predição *em float, antes de quantizar*, com modo por trecho de 64/256/1024 amostras – Lorenzo-1, Lorenzo-2, **regressão LSQ** (a reta viaja quantizada no stream; a predição não propaga o ruído das reconstruções – o truque do SZ3) e **sazonal/match model** (`p = rec[i−L] + rec[i−1] − rec[i−1−L]` , lag por SAD – repete o batimento anterior do ECG com correção de deriva).
4. **GOP (keyframe/delta)** – a cada N amostras vai um valor absoluto; cada bloco decodifica sozinho ⇒ seek real e perda limitada (herança do H.265/splat4d).
5. **Entropia** – v1–v3: zigzag varint + RLE de zeros + flate/gzip (v3 com dicionário preset treinado). v4: **range coder adaptativo carryless com dois estágios à escolha do canal** – alfabeto multi-símbolo por contexto, ou *bitplane* (flag de zero + sinal + magnitude bit a bit). Bitplane ganha onde zeros dominam (robótica −23%); alfabeto ganha em residuais densos (ECG).

Os seis formatos e o trade-off medido

FORMATO	MECÂNICA	SEEK	TRADE-OFF MEDIDO
v1 mono	varint do canal inteiro + gzip monolítico	não	o menor dos formatos v1–v3; baseline batch
v2 GOP	blocos independentes por keyframe, flate|stored por bloco, RLE, preditor hold/linear por bloco	**sim**	custa 1,2–2,8× vs v1 — o preço de perder o contexto entre blocos
v3 dict	v2 + dicionário treinado (flate preset-dict; zstd *perdeu* o sweep — header de frame caro em blocos pequenos)	**sim**	recupera 3–21% do custo do v2 mantendo o seek
v4 rc	predição em float por trecho + range coder de 2 estágios; batch, como SZ3	não	melhor batch em todos os casos; bate o SZ3 nos 15
v4 seek	preditores e range coder do v4 em blocos por GOP (modelos zerados por bloco; sazonal fica de fora)	**sim**	melhor formato *seekable* em todos os casos (bate o v3 por 1,2–2,3×); paga 1,3–8,3× vs v4 batch — modelos frios, keyframe absoluto e flush de 4 B por bloco
v5 xch	cross-channel: modelos de entropia compartilhados + predição inter-canal (diff contra referência decodificada); 3 layouts, fallback não-sólido	não	mocap −8%, carro prático −4%, robótica −3% vs v4; empate no resto (fallback garante ≤ v4); **−40%** na ablação de canais quase idênticos

Assimetria encode/decode — de projeto

Como em codec de vídeo, o decode é passada única e rápido; o encode paga a busca (2 estágios de entropia × 3 tamanhos de trecho × ~5 candidatos de modo simulados por trecho + procura de lag). Milhões de amostras/s (1 amostra = 1 float64), single-thread, Apple Silicon:

CASO	V1 ENC	V1 DEC	V2 ENC	V2 DEC	V4 ENC	V4 DEC	V4S ENC	V4S DEC
satélite	3,0	128,6	9,0	67,6	0,12	72,3	4,7	33,1
agentes	9,9	128,0	3,4	108,1	0,43	24,5	3,1	14,2
robótica	15,4	131,2	2,5	62,9	0,27	59,5	3,0	12,5
ECG sintético	1,3	134,2	4,9	24,8	0,08	29,0	2,7	14,3
ECG real	3,3	135,4	6,1	25,8	0,03	25,4	2,4	12,0

Decode do v4: 25–72 Ma/s ≈ 200–580 MB/s de dado reconstruído. Encode do v4 batch: 0,03–0,4 Ma/s — a busca fica explícita e medida; um encoder de produção a limitaria ou a faria adaptativa. O v4 seek codifica 10–80× mais rápido que o v4 batch (busca menor: sem variação de trecho, sem sazonal). v1/v2 enc incluem gzip/flate BestCompression; v1 dec não inclui gunzip.

4 Telecom: bytes no fio, perda, uplink e cifra

4.1 Transporte sem compressor: a coluna "s/entropia"

Nem todo link tem espaço para gzip. Um pacote NB-IoT (o cenário do caso satélite, que espelha a telemetria LEPTONSat via NB-IoT NTN) tem **~200 bytes** — o entropy coding de envelope não cabe no caminho. Por isso o benchmark reporta a coluna **s/entropia**: o payload varint+RLE puro, *antes* de qualquer flate/gzip. No satélite, esse payload cru já dá **45×** sobre o raw — "sensor parado ≈ zero bytes" vale literalmente no fio, não só depois do gzip. O preditor linear por bloco é onde mais paga exatamente aqui: payload s/entropia 6–12% menor em todos os casos (ECG real −12%, robótica −11%).

4.2 Bitrates efetivos medidos

Derivados por aritmética direta da tabela medida (formato **v4 seek**, o número da história de streaming — batch é ainda menor, mas sem seek):

CASO	DURAÇÃO	V4 SEEK	BITRATE EFETIVO	OBSERVAÇÃO
satélite (6 sensores @ 1 Hz)	24 h	21,4 KB	**≈ 0,25 B/s**	o dia inteiro de 6 sensores, com seek, em ~0,25 byte por segundo (~0,04 B/s por canal)
ECG real (MIT-BIH, 360 Hz)	30 min	273,7 KB	≈ 0,15 KB/s	ECG contínuo com garantia de ±1 LSB em ~1,2 kbit/s
robô real (ROS2, IMU ~203 Hz)	128 s	68,8 KB	≈ 0,54 KB/s	uplink de IMU handheld em movimento contínuo
manipulador (ALOHA, 14 juntas @ 50 Hz)	1.100 s	186,4 KB	≈ 0,17 KB/s	sessão de trabalho bimanual completa
mocap (62 DOFs @ 120 Hz)	88 s	237,5 KB	≈ 2,7 KB/s	esqueleto humano completo (batch v4: ~2,3 KB/s)
game (CS2, 50 canais @ 64 tick)	~258 s	156,3 KB	≈ 0,61 KB/s	replay visual dos 10 jogadores (±1 cm, ±0,1°)

Tamanhos: coluna "v4 seek" da tabela de resultados (seção 6). As durações vêm da descrição de cada dataset no README.

4.3 GOP = unidade de perda de pacote (e de seek)

Nos formatos v2/v3/v4s, cada bloco entre keyframes é auto-contido: `flag + len + bloco`. Consequências diretas para o projetista de link:

- **Perda limitada**: perder um datagrama corrompe no máximo o(s) GOP(s) que ele carregava — o próximo keyframe ressincroniza sozinho. Não há estado de sessão a reconstruir.
- **Seek real**: pular blocos pelo prefixo de tamanho, sem descomprimir — acesso aleatório no arquivo e ponto de entrada no stream.
- **Dimensionamento**: o tamanho do GOP é o botão que troca taxa por granularidade de perda. O custo medido do seek (v4s vs v4 batch, 1,3–8,3×) dói mais onde o GOP é curto (robótica: 60 amostras/bloco) — alinhar o GOP com o MTU/pacote do rádio é decisão de engenharia do link, não do codec.

4.4 Uplink robô→nuvem

O desenho de referência (plano de integração ROS2, seção 8): um nó em dual-write fecha janelas de N segundos em v4 seek — cada lote é um arquivo/mensagem auto-contido para o uplink, com perda limitada ao bloco. Na demo de frota (`demo/hover-labirinto.html`), poses ±2 cm/±0,5° sobem por dois uplinks medidos lado a lado: stream deadband/varint contra **TUBE v4 real em lotes de 10 s**, codificados, decodificados e verificados ao vivo — o v4 ganha ~30% do stream mesmo pagando modelos frios por lote; compartilhar ~25 células de obstáculo com a frota custa ~50 bytes de delta de mapa.

4.5 tubesec: envelope criptográfico (protótipo, `go run . -sec`)

Camadas para transmissão (ex.: downlink de satélite) — **cripto no container, nunca no codec**:

- **AES-256-GCM por GOP**: nonce derivado (não viaja); o AAD autentica caso/canal/**bound** — adulterar o ε quebra a tag, ou seja, o contrato de auditoria fica criptograficamente íntegro. Perda continua limitada ao GOP e o seek sobrevive à cifra. AES-256 já é pós-quântico para o payload (Grover → 2^{128}).
- **Acordo de chave híbrido X25519 + ML-KEM-768** por sessão: **2.304 B/rotação** (constantes FIPS 203).
- **Manifesto assinado ML-DSA-65**: **3.309 B/sessão** (FIPS 204) — a resposta certificada do índice (seção 5) vira evidência não-repudiável.

Roundtrip decifra→decodifica→bound ✓ nos 15 casos.

> **O achado medido — a tag de 16 B/GOP domina em compressão extrema.** O selo GCM é invisível onde o bloco é gordo (AMI +4,4%, ECG real +10,3%, mocap +17%) e **domina** onde a compressão é extrema com GOP curto: **robótica +172%, satélite +126%** — o GOP médio do satélite tem ~12 B de payload, *menor que a própria tag*. Ninguém tem esse problema a 2–5× de compressão; a 50–190× ele aparece. A resposta de engenharia é selar por **lote de GOPs** (o pacote de rádio), casando a granularidade da cifra com a unidade de perda — a lição geral: *granularidade da cifra = unidade de perda*.
>
> **Declarado e não medido — traffic analysis do deadband:** o tamanho do GOP vaza atividade mesmo cifrado (o próprio detector de surpresa do codec é o ataque). Padding é a mitigação conhecida, e custa taxa.

5 Índice por GOP e queries certificadas

O v4 seek dá acesso aleatório sem descomprimir; o passo seguinte é responder **"quando o canal passou de X?"** tocando só os blocos relevantes. Um índice **sidecar** (o stream v4s não muda um byte; o contrato de vetores Go⇄JS segue intacto) guarda o min/max da reconstrução de cada GOP, arredondado para a grade de quantização, em varint delta: **2–4 B por bloco** medidos — no satélite, 579 B de índice para um stream de 3,4 KB.

A query de intervalo aberto `(lo, hi)` herda o bound e sai **certificada**:

- **excluído com prova**: bloco cujo `[min−bound, max+bound]` não toca o intervalo é podado com prova comprimida de exclusão;
- **definite**: nos blocos decodificados, amostra cujo `rec±bound` cai inteiro dentro do intervalo — o original está dentro, GARANTIDO;
- **possible**: amostra a ±bound da borda — a fronteira honesta.

É o zone map do Parquet com ε como contrato — e com a distinção definite/possible que só um bound determinístico permite (sketches e amostragem dão erro probabilístico; aqui a resposta vem com garantia, não com confiança). Com o manifesto ML-DSA-65 do tubesec, a tricotomia certificada vira **evidência não-repudiável** — resposta forense com assinatura.

Poda medida (`go run . -query`: canal 0 de cada caso, threshold no p97, verificado contra o decode completo)

CASO / CANAL	BLOCOS TOCADOS	BYTES DECODIFICADOS	ÍNDICE	SPEEDUP
satélite `temp`	25/288	8%	579 B (2,0 B/GOP)	**12,8×**
mocap `root.0`	4/42	12%	115 B	8,8×
CS2 `p0_x` (visual)	3/64	11%	186 B	13,8×
carro `can_speed` (prático)	2/20	10%	42 B	6,4×
manipulador `left_waist`	15/215	9%	468 B	10,8×
robô ROS2 `gyro_x`	40/102	41%	271 B	2,3×
drone `gyro[0]`	61/67	91%	144 B	1,1×
medidor `kwh` (1 casa)	50/53	96%	161 B	1,0×
ECG real `mlii`	**1806/1806**	**99%**	5,3 KB	1,0×

A leitura honesta está nas três últimas linhas: **a poda só existe quando o evento é localizado**. Temperatura, posição e velocidade cruzam o p97 em poucos trechos do dia; o ECG cruza em TODO GOP (cada bloco de 1 s contém um pico R acima do p97 — a estrutura do sinal, não uma fraqueza do índice); giroscópio ruidoso e consumo semanal de uma casa idem. Nesses casos o índice não atrapalha (4 B/bloco, query ≡ decode completo); nos outros, responde tocando ~10% dos bytes. Canais circulares (`Period != 0`) ficam de fora: min/max não é bem definido num círculo. Coberto por teste de propriedade nos dois lados do contrato (Go e JS, mesma serialização): query com poda ≡ brute force. Demo interativa: `demo/indice-gop.html`.

6 Benchmarks completos

Seed fixa; ECG real do PhysioNet; 11 dos 15 casos com dados reais públicos. `vs raw (v4s)` usa o v4 seek – o número da história de streaming. `s/entropia` = payload varint+RLE antes do entropy coding. `deadband` = % de amostras não transmitidas. `maxerr/bound` ≤ 1,00 = garantia cumprida em toda amostra, verificada no decode.

6.1 Os 15 casos (formatos internos e baselines lossless)

CASO	RAW	GZIP(RAW)	GORILLA	V1 MONO	V2 GOP	V3 DICT	V4 RC	V4 SEEK	V5 XCH	S/ENTROPIA	VS RAW (V4S)	DEADBAND	MAXERR
satélite (24h@1Hz, 6 sensores)	4,0 MB	3,1 MB	3,0 MB	28,3 KB	62,8 KB	49,4 KB	**3,1 KB**	21,4 KB	3,1 KB	83,1 KB	**189×**	91,8%	1,00 ✓
agentes (50 agentes, 1h@5s)	843,8 KB	44,3 KB	66,2 KB	21,2 KB	24,9 KB	24,9 KB	**13,3 KB**	20,7 KB	13,5 KB	24,7 KB	**41×**	90,0%	1,00 ✓
robótica (200 corpos, 30s@60Hz)	11,0 MB	9,9 MB	9,7 MB	113,6 KB	315,7 KB	265,5 KB	**26,3 KB**	217,8 KB	25,5 KB	329,2 KB	**52×**	89,2%	1,00 ✓
ECG sintético (10min@250Hz, ±5µV)	1,1 MB	1,1 MB	1,2 MB	**33,6 KB**	68,1 KB	60,7 KB	33,8 KB	52,9 KB	33,8 KB	108,8 KB	**22×**	45,2%	1,00 ✓
ECG real (MIT-BIH 100, 30min@360Hz, ±5µV)	5,0 MB	626,0 KB	4,3 MB	246,5 KB	331,8 KB	325,9 KB	**208,2 KB**	273,7 KB	208,2 KB	608,2 KB	**19×**	33,6%	1,00 ✓
drone real (PX4, IMU+EKF+motores)	1,4 MB	508,3 KB	625,7 KB	18,9 KB	32,0 KB	30,0 KB	**6,5 KB**	12,1 KB	6,4 KB	50,0 KB	**123×**	86,2%	1,00 ✓
robô real (ROS2/MCAP, IMU Livox)	1,4 MB	441,0 KB	724,0 KB	66,0 KB	89,3 KB	89,3 KB	**50,5 KB**	68,8 KB	50,5 KB	124,3 KB	**21×**	49,2%	1,00 ✓
manipulador real (ALOHA, 14 juntas)	6,7 MB	671,0 KB	1,7 MB	171,3 KB	257,0 KB	243,6 KB	**124,8 KB**	186,4 KB	124,8 KB	420,3 KB	**37×**	67,1%	1,00 ✓
carro real (CAN, bound = passo nativo)	272,0 KB	87,2 KB	195,4 KB	21,2 KB	25,7 KB	25,7 KB	**20,0 KB**	22,5 KB	20,0 KB	31,5 KB	**12×**	15,4%	1,00 ✓
carro real (CAN, bound prático)	272,0 KB	87,2 KB	195,4 KB	7,4 KB	11,4 KB	11,4 KB	**5,1 KB**	7,5 KB	4,9 KB	17,9 KB	**36×**	68,5%	1,00 ✓
mocap real (CMU 86_02, 88s@120Hz)	5,0 MB	2,8 MB	4,8 MB	250,2 KB	294,8 KB	294,8 KB	**198,4 KB**	237,5 KB	182,4 KB	517,1 KB	**22×**	42,1%	1,00 ✓
medidor real (AMI, bound=passo nativo)	26,7 MB	5,8 MB	24,8 MB	3,8 MB	4,3 MB	4,3 MB	**3,4 MB**	3,7 MB	3,4 MB	4,3 MB	**7,2×**	3,2%	1,00 ✓
medidor real (AMI, bound prático)	26,7 MB	5,8 MB	24,8 MB	1,8 MB	2,2 MB	2,2 MB	**1,7 MB**	1,9 MB	1,7 MB	3,2 MB	**14×**	23,0%	1,00 ✓
game real (CS2 GOTV, replay visual)	6,2 MB	995,8 KB	1,1 MB	152,7 KB	192,2 KB	190,2 KB	**112,8 KB**	156,3 KB	112,9 KB	268,9 KB	**41×**	78,0%	1,00 ✓
game real (CS2 GOTV, bound perícia)	6,2 MB	995,8 KB	1,1 MB	285,9 KB	298,1 KB	296,0 KB	**210,0 KB**	265,4 KB	210,1 KB	328,5 KB	**24×**	70,0%	1,00 ✓

Gorilla = baseline lossless XOR de float64 (padrão de TSDB, implementada e testada no repo). v3 dict inclui o dicionário no total. A coluna "envelope" do README (vencedor do sweep flate preset-dict vs zstd) foi omitida por espaço: flate venceu em todos os casos.

6.2 Benchmark externo: SZ3 e ZFP (mesmo bound ABS por canal; garantia deles verificada por decode)

CASO	V1 MONO	V4 RC	V3 DICT (SEEK)	SZ3	MAXERR	ZFP	MAXERR
satélite	28,3 KB	**3,1 KB**	49,4 KB	13,9 KB	1,00 ✓	749,1 KB	0,58 ✓
agentes	21,2 KB	**13,3 KB**	24,9 KB	49,4 KB	1,00 ✓	67,5 KB	0,47 ✓
robótica	113,6 KB	**26,3 KB**	265,5 KB	180,4 KB	1,00 ✓	1,4 MB	0,73 ✓
ECG sintético	**33,6 KB**	33,8 KB	60,7 KB	44,3 KB	1,00 ✓	167,6 KB	0,46 ✓
ECG real (MIT-BIH)	246,5 KB	**208,2 KB**	326,6 KB	240,2 KB	1,00 ✓	861,5 KB	0,46 ✓
drone real (PX4)	18,9 KB	**6,5 KB**	30,0 KB	20,0 KB	1,00 ✓	145,1 KB	0,73 ✓
robô real (ROS2/MCAP)	66,0 KB	**50,5 KB**	89,3 KB	73,4 KB	1,00 ✓	244,3 KB	0,58 ✓
manipulador real (ALOHA)	171,3 KB	**124,8 KB**	243,6 KB	176,2 KB	1,00 ✓	1,1 MB	0,57 ✓
medidor real (AMI nativo)	3,8 MB	**3,4 MB**	4,3 MB	6,0 MB	1,00 ✓	6,0 MB	0,61 ✓
medidor real (AMI prático)	1,8 MB	**1,7 MB**	2,2 MB	2,1 MB	1,00 ✓	4,4 MB	0,47 ✓
carro real (CAN nativo)	21,2 KB	**20,0 KB**	25,7 KB	29,7 KB	1,00 ✓	67,7 KB	0,52 ✓
carro real (CAN prático)	7,4 KB	**5,1 KB**	11,4 KB	8,0 KB	1,00 ✓	51,2 KB	0,52 ✓
mocap real (CMU 86_02)	250,2 KB	**198,4 KB**	294,8 KB	277,6 KB	1,00 ✓	893,2 KB	0,45 ✓
game real (CS2, replay visual)	152,7 KB	**112,8 KB**	190,2 KB	177,8 KB	1,00 ✓	1,0 MB	0,58 ✓
game real (CS2, bound perícia)	285,9 KB	**210,0 KB**	296,0 KB	394,4 KB	1,00 ✓	1,3 MB	0,47 ✓

Comparação justa é contra o v1 mono/v4 (batch, sem seek, como eles). O ZFP não é competitivo em série temporal 1D ruidosa (7–50× pior): é desenhado para campos suaves multidimensionais, e o modo accuracy arredonda a tolerância para potência de 2 – o maxerr ~0,5 mostra que entrega o dobro da precisão pedida e paga bits por isso. O que eles não dão neste modo: streaming incremental, seek por GOP e perda limitada – comparando o v3 (que dá as três) com o SZ3, paga 1,4–3,6× nos casos que o SZ3 vence e ainda ganha por 2,0× nos agentes.

6.3 SZ3 afinado (`go run . -sz3tune`) – a ressalva "defaults" fechada

O placar foi re-medido dando ao SZ3 as mesmas vantagens do encoder do TUBE: **best-of de algoritmo por canal** (interp_lorenzo/interp/lorenzo_reg) e, nos casos de bound uniforme, o **layout 2D canais×tempo nas duas orientações** (o habitat multidimensional dele). Resultado: **15 de 15 sobrevive**. O tuning move onde havia o que mover – satélite −37,9% (margem 4,5×→2,8×), drone −20% (3,1×→2,5×), robótica −16% – e não move nada nos ECGs (0%: defaults já ótimos). O 2D é a arma real no caso certo: AMI nativo 6,0→3,9 MB (−35%), estreitando a margem para **1,15×, a menor do placar** (empatada com o ECG real); no AMI prático o 2D não ajuda. Garantia do SZ3 verificada em todas as execuções, inclusive 2D.

6.4 Mercado: swinging door (`go run . -market`) – o algoritmo dos historians

SDT nos mesmos canais com o mesmo bound ABS, na variante com garantia de verdade (endpoint testado contra as portas – a variante ingênua admite até 2× o bound e *falhou a verificação*); tamanho contado a favor dele: min(cru, gzip) dos pivôs.

CASO	TUBE V4	V4 SEEK	SWING DOOR	SDT/V4	PIVÔS RETIDOS
satélite	**3,1 KB**	21,4 KB	29,6 KB	9,6×	0,7%
ECG sintético	**33,8 KB**	52,9 KB	341,9 KB	10,1×	27,5%
ECG real	**208,2 KB**	273,6 KB	526,7 KB	2,5×	54,9%
drone real	**6,5 KB**	12,1 KB	39,3 KB	6,1×	4,2%
mocap real	**198,4 KB**	237,5 KB	1,6 MB	8,1×	43,2%
CS2 visual / perícia	**112,8 / 210,0 KB**	156,3 / 265,4 KB	516,3 / 826,5 KB	4,6× / 3,9×	16 / 28%
manipulador ALOHA	**124,8 KB**	186,4 KB	456,6 KB	3,7×	26,6%
robô ROS2	**50,5 KB**	68,8 KB	245,1 KB	4,9×	43,2%
medidor AMI nativo / prático	**3,4 / 1,7 MB**	3,7 / 1,9 MB	6,1 / 5,4 MB	1,8× / 3,3×	97 / 79%
carro CAN nativo / prático	**20,0 / 5,1 KB**	22,5 / 7,5 KB	86,9 / 42,4 KB	4,4× / 8,4×	82 / 25%

TUBE ganha do proxy de historian por 1,8×–10,1× nos 15 casos (robótica e agentes sintéticos confirmam: 9,7× e 4,0×) – e até o v4 seek ganha em todos. O porquê estrutural: o SDT paga ≥9 bytes por pivô retido (índice + float64 cru), enquanto o residual pós-predição do TUBE custa frações de bit – no satélite o SDT retém só 0,7% dos pontos e AINDA fica 9,6× maior. Onde o sinal não deixa reter pouco (AMI nativo: 97% retido), o SDT quase não comprime. Garantia do SDT verificada na reconstrução linear (✓ nos 15).

6.5 ModelarDB e BUFF (os vizinhos de contrato, rodados de verdade — seleção por caso)

CASO	TUBE V4	MODELARDB	BUFF
satélite	**3,1 KB**	16,3 KB ⚠	791,1 KB
ECG real	**208,2 KB**	683,7 KB	872,8 KB
drone real	**6,5 KB**	22,2 KB	233,0 KB
mocap real	**198,4 KB**	1,0 MB	872,2 KB
CS2 perícia	**210,0 KB**	694,9 KB	1,6 MB
manipulador ALOHA	**124,8 KB**	1,2 MB ⚠	1,3 MB
robô ROS2	**50,5 KB**	159,6 KB ✗	259,1 KB
AMI nativo	**3,4 MB**	7,3 MB	5,3 MB
carro CAN prático	**5,1 KB**	32,0 KB ✗	42,6 KB

Harness Rust sobre os mesmos canais/bounds (`--dump` + crates deles): ModelarDB-RS (crate `modelardb_compression`, API `ErrorBound::try_new_absolute`) e BUFF (encoder canônico `byte_fixed` extraído; bound→precisão p = ⌈log10(0,5/bound)⌉). **TUBE v4 vence os dois em todos os 15 casos** (ModelarDB 2,1×–9,5×; BUFF 1,55×–255×). ✗/⚠: o ModelarDB **viola o próprio bound** — é f32 interno e, em canais de valores grandes (timestamps epoch), o erro contra o f64 original chega a $2{,}55\times10^{8}$ vezes o bound (✗ = violação grossa; ⚠ = 1,3–1,8×; nos ✗ os bytes saem ainda subestimados). Nos demais ~1.576 canais o bound valeu (conferido por decode). O BUFF paga a grade decimal (ε só em potências de 10 força sobre-precisão — satélite: GPS a p5 → 255× pior) e a cegueira temporal (robótica 76× pior) — o preço de ser consultável in-situ. Caveats de contagem todos a favor DELES (ModelarDB sem coluna de erro/tags/overhead Parquet; BUFF é o buffer completo, bound verificado nos 15).

7 Limites honestos e hipóteses refutadas

A marca do projeto é registrar o que *não* funcionou com o mesmo cuidado dos ganhos. Isto não é apêndice — é o que torna os números da seção 6 críveis.

- **AMI: a estimativa refutada pela medição.** A hipótese pré-medição ("30–80×, como o satélite") errou o gêmeo: kWh meio-horário de UMA residência é ruidoso — geladeira cicla, chuveiro liga, o valor muda a cada intervalo (deadband de só 3,2% no bound nativo, 23% no prático). Medido: 7,2× e 14× — regime ECG/carro-em-movimento, não satélite. A margem sobre o SZ3 afinado é a menor do placar (1,15× no nativo com layout 2D). A metade da tese AMI que segue *não medida* é a dos canais suaves (tensão/corrente/FP, que o dataset não tem) — e a query DRP/DRC de tensão que motivou o caso continua sendo aposta não testada.
- **v5 no manipulador: hipótese refutada.** A cadeia cinemática do ALOHA parecia o caso ideal de cross-channel; o resultado foi empate exato com o v4 (fallback não-sólido). Dois braços em sub-tarefas distintas não co-movem como a marcha do mocap (−8%): a correlação existe na *estrutura*, mas não sobra redundância explorável nos *residuais*. O mesmo no CS2 (jogadores não co-movem) e no robô ROS2 (eixos de IMU).
- **ECG real: o limite físico.** Com bound = ±1 LSB do ADC, o stream é dominado por ruído do sensor — nenhum preditor comprime ruído (o sazonal é escolhido em 16% dos trechos e não move o total). O ganho para em 19× e o empate técnico com o SZ3 (1,15×) é o teto do sinal, não do codec. Versão automotiva do mesmo limite: CAN com bound = ½ passo nativo dá 12×; com bound prático de frota, 36–53× — quem escolhe o bound escolhe o ponto entre perícia e taxa (resposta de política, não de código).
- **Encode lento — por projeto.** 0,03–0,4 Ma/s no v4 batch contra 25–72 Ma/s de decode: a busca exaustiva de modos fica explícita e medida. SZ3 segue ganhando em velocidade de encode e maturidade; a alegação do TUBE é taxa com garantia, não throughput de encode.
- **A poda do índice não é universal.** ECG 1806/1806 blocos tocados (speedup 1,0×), giroscópio de drone 91%, consumo de uma casa 96%: threshold em sinal que cruza o limiar em todo bloco não poda — é a estrutura do sinal.
- **Tag GCM em compressão extrema.** +126% no satélite, +172% na robótica (seção 4.5): o tubesec por GOP não é usável como está nesses regimes; a resposta (selar por lote) está desenhada, não medida. Traffic analysis do deadband: declarado e não medido.
- **Caveats permanentes:** bounds escolhidos com folga realista mas não validados com engenheiro de domínio; dicionário do v3 treinado nos mesmos dados que comprime (num deploy real, treino offline em histórico da mesma família — ganho tende a ser menor); o robô ROS2 é um handheld de SLAM que gira o tempo todo — um robô de armazém deve ter deadband bem maior, número a medir com bag de cliente.

8 Integração

8.1 Contrato Go ⇄ JS (o codec das demos é o mesmo codec)

O decodificador existe em Go puro (stdlib; a única dependência externa é usada no treino de dicionário offline e no candidato zstd) e em JS (`demo/codec.js` , `demo/codec4.js` — porte byte a byte do v4/v4 seek). O contrato entre as implementações é de **bytes idênticos**, congelado em vetores:

- `go run . -vectors` gera `demo/test-vectors.json` (encode em hex por bloco + reconstrução exata esperada); `node demo/codec.test.js` e `demo/codec4.test.js` validam que o JS reproduz bytes e floats exatamente, mais fuzz; `TestVectorsUpToDate` (Go) falha se o encoder mudar sem regenerar o JSON.
- O contrato exigiu duas correções de ponto flutuante que valem registro: o `Math.round` do JS é `floor(x+0.5)` (arredonda duas vezes — o JS implementa o half-away-from-zero do Go via `x-trunc(x)` , exato por Sterbenz); e o compilador Go funde `v - float64(last)*step` em FMA no arm64, divergindo do JS no ½ passo exato — o encoder força o arredondamento intermediário com `float64(...)` explícito. Detalhe de implementação que vira requisito de interoperabilidade: quem portar o codec para C++/Rust vai reencontrar os dois.

8.2 ROS2 / MCAP: o `rosgate`

O caso "robô real" já roda sobre rosbag2/MCAP nativo: o `tools/rosgate` é um leitor MCAP mínimo (magic + registros + chunks zstd, ~200 linhas, sem dependência nova) com decodificador CDR de `sensor_msgs/msg/Imu` na mão. É o mesmo padrão dos leitores ULog (PX4), CAN (comma2k19), AMC (mocap) e npy do repo: poucas centenas de linhas auditáveis por formato de entrada. A forma alvo da CLI (degrau 1 do plano de adoção):

```
rosgate archive missao.mcap -bounds bounds.yaml -o missao.tube   # comprime + indexa
rosgate query missao.tube "joint3.effort > 12" --certified        # forense com prova
rosgate verify missao.tube missao.mcap                            # confere o bound, canal a canal
```

O `bounds.yaml` é versionado pelo cliente: **o ε é decisão de engenharia de quem conhece o robô, não do codec.**

8.3 Degraus de adoção 0–3 (reversíveis; nenhum toca o caminho de controle)

#	O QUE É	ESFORÇO DO CLIENTE	GATE PARA O PRÓXIMO DEGRAU
`0`	POC com um rosbag que já existe (10–30 min de operação + bounds assinados por um engenheiro + uma pergunta forense real)	enviar 1 arquivo	taxa ≥ alvo combinado vs MCAP+zstd e query de exceedance funcionando no dado deles
1	CLI offline `rosgate` (read-only) sobre os próprios bags	rodar um binário	engenheiros usam sem fricção; `verify` os convence do contrato
2	nó `tube_bridge` em dual-write: 1 robô, 30 dias, em paralelo com o rosbag atual (rollback = remover do launch)	1 robô, 30 dias	economia medida + zero incidente + reconstrução auditada contra o bag bruto
3	retenção de produção: bruto vive 7–30 dias; longo prazo vira TUBE + índice (consultável, com garantia declarada); plugin C++ do rosbag2 opcional	mudança de política de retenção	decisão de negócio com os números do degrau 2

O princípio do plano: cada degrau é reversível, tem critério de decisão numérico combinado antes, e o seguinte só acontece se o anterior provar valor nos dados do cliente. O tempo real não passa pelo TUBE em nenhum degrau.

8.4 Referências

- Repositório (código, benchmark reproduzível, demos e testes de contrato): `github.com/andradeandrey/telemetria`
- Relatório completo da evolução: `docs/relatorio.pdf` · Glossário de siglas e termos: `docs/GLOSSARIO.md` / `docs/glossario.pdf`
- Demo interativa do índice certificado: `demo/indice-gop.html` · demos de frota e replay 4D: `demo/hover-labirinto.html`, `demo/labirinto-4d.html`, `demo/robo-galpao-4d.html`
- Plano de integração ROS2 em degraus: `docs/plano-instor-ros2.md`
- Contexto de robótica e teleoperação: `stickybit.com.br/robotica/`
- Datasets reais (todos públicos): MIT-BIH/PhysioNet (ODC-BY), PX4 pyulog (BSD-3), comma2k19 (MIT), CMU MoCap, Low Carbon London (CC-BY 4.0), FAST-LIVO/MCAP (CC-BY-NC 4.0, uso de benchmark de pesquisa), lerobot/aloha_static_coffee (MIT), demos CS2 do demoinfocs-golang (MIT)
- Baselines: SZ3 (szcompressor), ZFP (LLNL), Gorilla (VLDB 2015), Swinging Door Trending, ModelarDB-RS, BUFF (PVLDB 2021)