CERWU

Compression with Entropy-Regularized Weight Updates

Alexander Conzelmann | Friday Talks 06.03.2026

Problem Setting

Models are HUGE

Even simple ResNets are hundreds of MB uncompressed

Edge Devices are Small

Sensors, phones & embedded systems have tiny storage & bandwidth

Privacy & Latency Matter

On-device inference avoids cloud round-trips and data sharing

Goal: Shrink stored model size as much as possible with minimal accuracy loss

Quantization ≠ Compression

Quantization (current methods)

- Each weight stored at fixed bit-width (e.g. 4 or 8 bits per weight)
- Great for inference speed
- Wastes bits on easy-to-predict weights

\+ entropy coding

Compression (Our Goal)

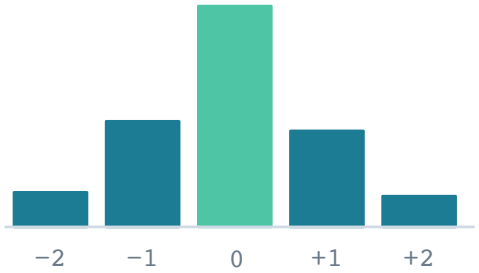
- Minimize total bit string length using entropy coding
- Effective bit-width varies per weight — even below 1 bit!
- Better compression = fewer bits for predictable weights

Entropy Coding – saving storage for free

Assign shorter bit strings to frequent symbols, longer ones to rare symbols

Step 1 — Build a Probability Model

Most weights cluster near zero — we know this in advance



Step 2 — Assign Bit Strings

Value	P(w)	Code
0	50%	0
−1	20%	100
+1	20%	101
−2	5%	1100
+2	5%	1101

Most frequent value 0 → just 1 bit!

Step 3 — Compress Data

Tensor: 0 0 0 +1 0 −1 0 0

BEFORE (fixed 3-bit storage):

000 · 000 · 000 · 001 · 000 · 101 · 000 · 000

= 8 × 3 bits = 24 bits

AFTER (entropy coded):

0 · 101 · 0 · 0 · 100 · 0 · 0

= 12 bits (50% saving!)

The Key Insight

Naive approach: quantize first → compress later.

Problem: weights W are quantized without awareness of later compression

Our approach: jointly optimize for accuracy AND bit rate.

Objective: $\min_{\hat{W}} ||W - \hat{W}||_2^2 + \lambda R(\hat{W})$

$D(\hat{W})$ = layer output error (how much accuracy we lose) $R(\hat{W})$ = information content (bits per weight)

How CERWU Works

1

Quadratic Rate Estimation: $R(\hat{W}) \approx \frac{\lambda\gamma}{2} || \hat{W} ||_2^2$

Approximate the (complicated) bit-rate term $R(\hat{W})$ by assuming a gaussian weight distribution. This makes the objective tractable for the weight update step (3).

2

Rate-Aware Quantization: $q(W_{ij}, \lambda, \gamma, P) = \text{argmin}_{g \in G} \left[\frac{(W_{ij} - g)^2}{2(H^{-1})_{jj}^2} - \lambda R(g) - \frac{\lambda\gamma}{2} g^2 \right]$

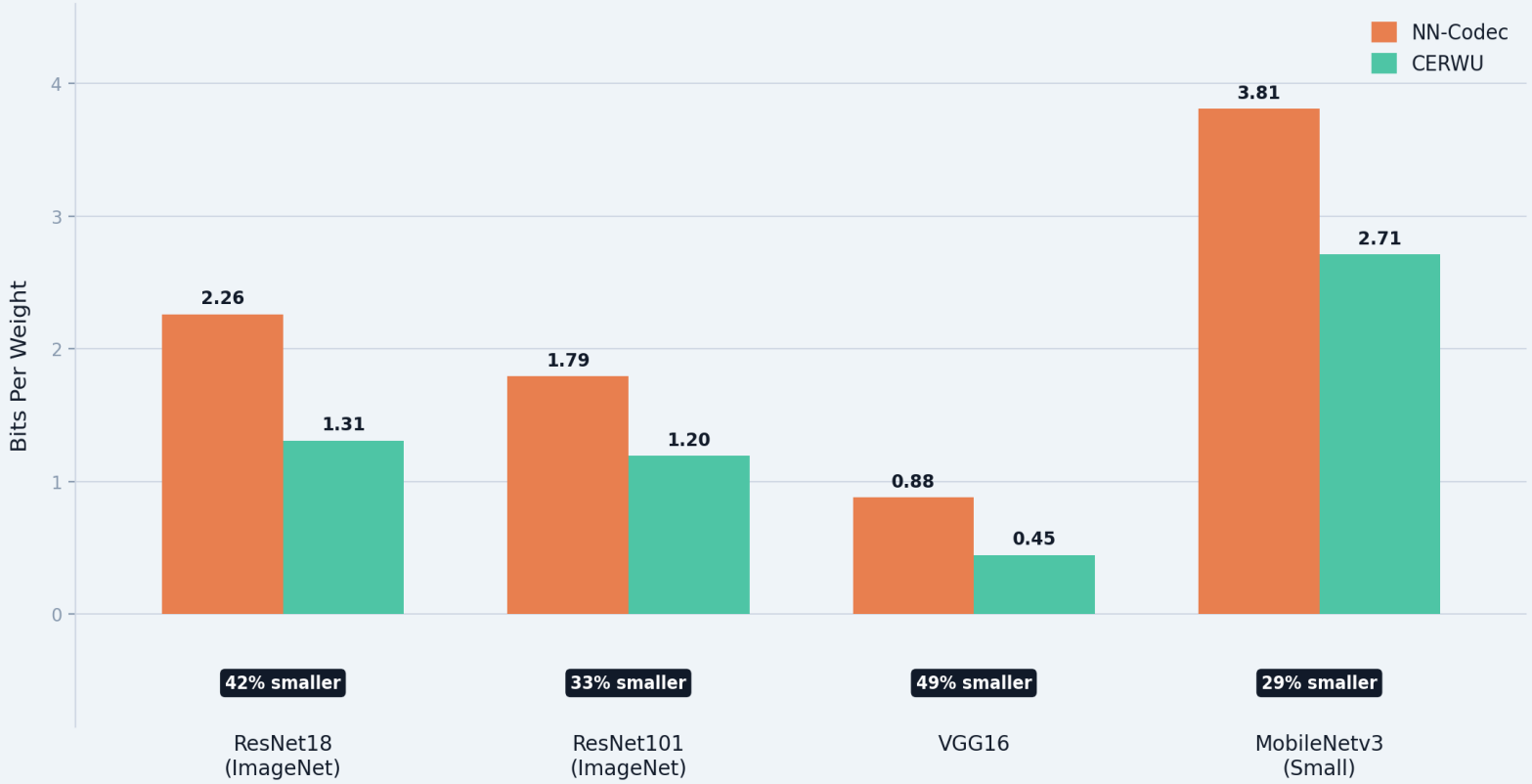
Iterate over fixed grid G (e.g. all 4-bit integers). Hessian H takes sensitivity into account.

3

Correct remaining weights after each quantization step

After quantizing each weight, update remaining weights in the same row to compensate for error (based on GPTQ adapted for our joint objective).

Results


NN-Codec
CERWU
Bits Per Weight
0
1
2
3
4
2.26
1.31
1.79
1.20
0.88
0.45
3.81
2.71
42% smaller
33% smaller
49% smaller
29% smaller
ResNet18 (ImageNet)
ResNet101 (ImageNet)
VGG16
MobileNetv3 (Small)

Practical Performance

<1s to decompress

Highly performant decoder

20-40% smaller

Compared to ISO neural network coding standard NNCodec

Strong adaptability

Quantization grids and entropy models are freely choosable

Smooth R/D Tradeoff

λ allows to choose between performance and compression

Limitations & Future Directions

Limitations

Transformers / LLMs show less promising results

Focus is on convolutional vision models, LLMs might need special handling

Encoding is slow for larger grids

Grid search over all quantization levels is expensive for large grids. Optimization (early stopping, etc.) could help.

Future Work

On-the-fly GPU decompression

Could reduce RAM↔GPU bandwidth during inference, saving energy and latency.

Different entropy models or grids

Plug in any autoregressive model; NLP-specific models may improve LLM compression.