PRESTO: PERSPECTIVE-BASED ENTROPY-SPARSITY TRAINING OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Pruning and quantization make a neural network smaller, and when the goal is storage size, a lossless compressor is eventually applied (Han et al., 2016) to a model that is usually not trained to be compressible.

We introduce PRESTO, a quantization-aware training method that adds to the loss a proxy for the compressed size of the weights, leaving the activations in floating point. This proxy is the value of a convex inner problem over the quantization grid learned by LSQ (Esser et al., 2020), written in perspective form, i.e. the tightest convex relaxation of the decision to keep a weight. A Lagrangian relaxation makes that problem separable, and an envelope argument gives a (sub)gradient of its value. To our knowledge, PRESTO is the first method to jointly balance sparsity and the codebook symbol distribution arising from quantization within a single convex inner problem, rather than through separate terms that act independently, as in HEMP+LOBSTER (Tartaglione et al., 2021) and LilNetX (Girish et al., 2023).

On ImageNet, with sizes measured on the whole checkpoint after lossless coding, PRESTO reaches $20.4\times$ on ResNet-18, $19.5\times$ on ResNet-50, $22.6\times$ on ViT-B/16, $11.0\times$ on DeiT-Small and $9.9\times$ on EfficientNet-B0, above full-precision accuracy on the two ResNets and less than 0.75 accuracy points below it on the other three; on five of the six networks, we hold an operating point that no published method we compare against betters on both axes. On AlexNet, prune-first methods remain ahead, a deficit we attribute to sparsity allocation.

1 INTRODUCTION

Model compression works through three complementary mechanisms: pruning removes parameters, quantization represents the surviving ones with a finite codebook, and lossless coding removes the statistical redundancy left in the resulting stream of codebook symbols. The classical Deep Compression pipeline applies the three in sequence (Han et al., 2016), and what it finally writes is a record of which weights survive and a stream of symbols for those that do, both then losslessly coded. Sparsity decides the cost of that record, the symbol distribution the cost of the stream, and the pipeline sets the two in separate stages.

Later methods differ in whether and when they control these costs: a first group acts on quantization, either during training or after it. Quantization-aware training (QAT) simulates low-precision arithmetic while the network is optimized, as PACT (Choi et al., 2018) and learned step size quantization (LSQ) (Esser et al., 2020) do, and joint methods such as CoDeQ (Wenshøj et al., 2025) also prune inside the quantizer by learning its dead zone (i.e., the interval mapped to zero), whereas mixed-precision methods such as HAWQ (Dong et al., 2019) choose bit widths from second-order information, and post-training methods such as BRECQ (Li et al., 2021) quantize an already trained network without retraining it. All of them control weight distortion or numerical precision, but none of them controls the entropy-coded size of the model, which we also call its code length. Two models that use the same codebook can produce symbol streams of very different distributions, and hence of very different code length (Table 1), and quantization-aware training that ignores code length may increase it as the network trains (Appendix C.1).

A second group targets the model storage size, but only once the parameters have been fixed. DeepCABAC combines rate-distortion quantization with context-adaptive coding (Wiedemann et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2019a), and it has since been standardized as ISO/IEC 15938-17 and implemented in the open-source, standard-compliant NNCodec (Becking et al., 2023), against which we compare our proposal. More recent post-training methods, such as L^2 (Shi et al., 2023) and the rate-constrained quantization of Conzelmann and Bamler (2025), follow the same approach. Code length is therefore an established target; however, the methods in this second group receive pre-trained parameters, so they can compress them but not change them in order to achieve better compressibility.

A third group incorporates model storage size itself into the training objective, together with quantization, as entropy-constrained training (Wiedemann et al., 2019b) and soft weight-sharing (Ullrich et al., 2017) do. These methods act mainly on the codebook symbol distribution, through an entropy penalty, and only some of them also target sparsity, as a separate term or prior. Specifically, entropy-penalized reparameterization (EPR) (Oktay et al., 2020) learns the parameters in a latent space with its own probability model and entropy penalty, without any sparsity term, and on ResNet-18 it gives the smallest known entropy-coded model without loss of accuracy, at 4.22% of the size of the uncompressed network. LilNetX (Girish et al., 2023) targets sparsity by applying separate penalties to structured and unstructured sparsity within a single formulation, achieving an even smaller size of 2.16% on the same ResNet-18, but 1.2 accuracy points below its own full-precision accuracy. The closest method to our approach is HEMP (Tartaglione et al., 2021), which adds a differentiable entropy term to the network's training loss and obtains sparsity from an independent pruning method, LOBSTER (Tartaglione et al., 2022), which shrinks and prunes parameters of low loss sensitivity. EPR and LilNetX store latent codes that a learned decoder expands into full-precision parameters, whereas our file holds a support mask marking the non-zero weights, integer codebook indices and one step size per tensor, from which the deployed weights follow by a scalar multiplication.

The methods reviewed above leave one question open: can sparsity and the symbol distribution be controlled inside one single objective while the network trains, rather than in separate stages run one after the other (Han et al., 2016), or by separate terms that each act on their own (Tartaglione et al., 2021; Girish et al., 2023)? What makes the question hard is that such an objective couples every weight of a tensor: the code length depends on the symbol counts, which are sums over the whole tensor, so the cost of assigning one weight to a symbol depends on how many others have chosen it. The problem therefore does not separate across weights the way a magnitude or ℓ_2 penalty does.

We propose a method, called PRESTO (which stands for PeRspective-based Entropy–Sparsity Training Optimization), that addresses the research question above by taking into account quantization, sparsity and compression within a single objective. While the network trains, PRESTO assigns the weights to the codebook symbols and learns the grid's step sizes, so that more weights fall on the zero symbol, which increases sparsity and shortens the code, while, in the configurations we report, the symbols of the weights mapped to non-zero symbols also become more concentrated. To achieve this, PRESTO adds to the training loss a proxy for the code length, defined as the value of a convex inner problem that assigns each weight to the codebook symbols. That inner problem inherits the coupling described above: PRESTO writes it in perspective form, the tightest convex relaxation of the decision to keep a weight, and removes the coupling with a Lagrangian relaxation of the count constraints. Perspective reformulations have already produced effective regularizers for structured pruning (Cacciola et al., 2023), where the decision they relax is binary. Carrying that form over to a relaxed assignment of each weight across codebook symbols is new as far as we know. Being the tightest such relaxation is what the term contributes: it prices an undecided weight higher than a linear coupling would, while leaving a weight that is clearly kept or clearly dropped at the cost the discrete problem gives it (Section 2.2). The resulting subproblems are solved exactly and without iteration, and return envelope derivatives for the latent weights and the learned step sizes (Section 3). PRESTO is thus the first method we are aware of in which sparsity and the symbol distribution are balanced inside one convex inner problem, with one value function and one vector of multipliers per tensor.

What distinguishes PRESTO from HEMP+LOBSTER and LilNetX is not how many terms they use, since PRESTO uses several too, but how those terms are combined. In HEMP+LOBSTER and LilNetX, they are separate terms, each acting on its own: in LilNetX all of them are penalties added to the loss, a rate term and two sparsity penalties on the same latents, each weighted by its own coefficient; in HEMP the entropy term is added to the loss, while LOBSTER prunes through its own update rule. Here, they are the objective of one constrained inner problem, solved exactly at a fixed multiplier instead of being smoothed, so the inner solver settles the trade-off before anything reaches the network, which receives one subgradient of the value function rather than a sum of separate

gradients. PRESTO also creates zeros by widening a dead zone, as CoDeQ does, but through the learned step sizes rather than a dedicated parameter.

We evaluate PRESTO on six ImageNet networks, against published methods and against its own variant, named the *LSQ baseline*, in which the regularizer is switched off and which is trained for as many epochs as the PRESTO run it is compared with, unless stated otherwise (Table 3). PRESTO roughly halves the packet, the coded file, of that baseline on both ResNets while its accuracy stays above the full-precision checkpoint, on ViT-B/16 it is five accuracy points above the standardized codec NNCodec at a smaller size, and on ResNet-18 it ties with EPR, the strongest published point; Tables 1 and 6 and Figure 1 give the points. On AlexNet, prune-first methods remain ahead, a deficit we attribute to sparsity allocation and discuss in Section 4.4.

2 The PRESTO Method

PRESTO trains a quantized network with one additional term in the loss. LSQ provides the quantization levels of each tensor; an inner problem decides how much of each weight is represented by non-zero symbols and on which symbols, trading sparsity against the symbol distribution; a Lagrangian relaxation makes that decision separable across weights and symbols; and the envelope derivatives of its value move both the latent weights and the step sizes. After training, the mask of the surviving weights and the stream of their symbols are losslessly coded. Section 2.1 fixes the quantizer, Section 2.2 defines the inner problem and Section 3 solves it; Appendix A gives every derivation in full.

2.1 Setting, Notation and the Quantizer

Consider a network whose parameters θ divide into the weight tensors that we quantize, indexed by Λ, and the remaining floating-point parameters θ_{fp}, which are the biases, the normalization parameters and, on the transformers, the class token and the positional embeddings. Tensor $\ell \in \Lambda$ has n_ℓ entries indexed by $i \in \mathcal{I}_\ell$ and latent weights $w^{(\ell)}$, and tensor ℓ uses a codebook of cardinality C_ℓ, of which $\mathcal{B}$ denotes the $B_\ell := C_\ell - 1$ non-zero symbols; we drop the subscript wherever the tensor is clear from context. We keep the zero symbol separate from that set, because the relaxation of Section 2.2 represents it by a mass $z_i^{(\ell)}$ instead of by an element of $\mathcal{B}$, so that the same zero value never has two representations. The task loss $\mathcal{L}_{\mathrm{task}}$ is the cross-entropy risk, to which four of the six networks add a distillation term (Appendix B.1), and Appendix D collects the notation.

We build on LSQ (Esser et al., 2020), in which each tensor carries one positive step size a_ℓ learned together with the weights. A latent weight is mapped onto the integer grid by $r_i^{(\ell)} := w_i^{(\ell)}/a_\ell$, then clipped to the representable range $[Q_n, Q_p]$ of the codebook, rounded and scaled back, so that the deployed value is $\widehat{w}_i^{(\ell)} = a_\ell \left\lfloor \mathrm{clip}(r_i^{(\ell)}, Q_n, Q_p) \right\rceil$ and the clipped latent weight is $\bar{w}_i^{(\ell)} := a_\ell \,\mathrm{clip}(r_i^{(\ell)}, Q_n, Q_p)$. The rounding is to nearest with ties to even and is handled in the backward pass by a straight-through estimator. With that convention the deployed value is the zero symbol exactly when

$$\left| w_i^{(\ell)} \right| \le \tfrac{1}{2} a_\ell, \tag{1}$$

so that every zero in our results is produced by that rule and by no external pruning step (in the per-channel runs of the appendix zeros are also counted with a magnitude threshold of 10^{-3}, as noted in Table 11; no run prunes a weight). Appendix A.1 gives the grid and the codebook values $v_b^{(\ell)} = a_\ell q_b$ in full, and Appendix B.1 the initialization of a_ℓ.

2.2 Perspective and Entropy-Based Regularizer

The letter b indexes an element of $\mathcal{B}$, and q_b is the corresponding integer code. In tensor ℓ, the real-valued level associated with symbol b is

$$v_b^{(\ell)}(a_\ell) := a_\ell q_b. \tag{2}$$

For every pair $(i, b) \in \mathcal{I}_\ell \times \mathcal{B}$ we introduce a non-negative assignment mass $x_{i,b}^{(\ell)}$, and write $y_i^{(\ell)} \in [0, 1]$ for the total mass with which weight i is represented by non-zero symbols, $z_i^{(\ell)} := 1 - y_i^{(\ell)}$

for the mass on the zero symbol, and $c_b^{(\ell)} \geq 0$ for the relaxed count of symbol b. The mass $z_i^{(\ell)}$ is a relaxed variable and not a hard pruning decision: it enters the PRESTO subgradient but never overwrites a latent weight with zero. Below, a_ℓ and $\bar{w}^{(\ell)}(a_\ell)$ are arguments while $x^{(\ell)}, y^{(\ell)}, c^{(\ell)}$ are the variables of the inner minimization, and $T_1, T_2, T_3 \geq 0$ are the three coefficients. We define[1]

$$\Phi_\ell(\bar{w}^{(\ell)}, a_\ell) := \min_{x^{(\ell)}, y^{(\ell)}, c^{(\ell)}} \quad T_1 \sum_{i \in \mathcal{I}_\ell} \frac{(\bar{w}_i^{(\ell)})^2}{y_i^{(\ell)}} + T_2 \sum_{i \in \mathcal{I}_\ell} y_i^{(\ell)} + T_3 \sum_{b \in \mathcal{B}} c_b^{(\ell)} \log_2 c_b^{(\ell)} \tag{3}$$

$$\text{s.t.} \quad \sum_{b \in \mathcal{B}} v_b^{(\ell)}(a_\ell) x_{i,b}^{(\ell)} = \bar{w}_i^{(\ell)}, \qquad i \in \mathcal{I}_\ell, \tag{4}$$

$$\sum_{b \in \mathcal{B}} x_{i,b}^{(\ell)} = y_i^{(\ell)}, \qquad i \in \mathcal{I}_\ell, \tag{5}$$

$$c_b^{(\ell)} = \sum_{i \in \mathcal{I}_\ell} x_{i,b}^{(\ell)}, \qquad b \in \mathcal{B}, \tag{6}$$

$$x_{i,b}^{(\ell)} \geq 0, \quad 0 \leq y_i^{(\ell)} \leq 1, \qquad i \in \mathcal{I}_\ell,\ b \in \mathcal{B}. \tag{7}$$

The T_1 term uses the closed extension of u^2/y: it equals u^2/y for $y > 0$, is 0 at $(u, y) = (0, 0)$ and $+\infty$ when $y = 0$ with $u \neq 0$, which makes explicit that a weight can carry zero mass only when the weight itself is zero. The three coefficients then play different roles. T_1 weights this term, which we call the perspective ridge. T_2 penalizes the mass assigned to non-zero symbols and is the sparsity coefficient, since that mass is what the support mask records. T_3 is the one that acts on the symbol distribution, and weights what we call the entropy term, by which we mean the convex penalty $\sum_b c_b^{(\ell)} \log_2 c_b^{(\ell)}$ on the relaxed counts and nothing more. By the symbol distribution we mean the counts $c_b^{(\ell)}$ of the non-zero symbols, on which the entropy term is defined; the term does not lower the entropy of those counts. What it shortens is the ideal zero-order code length of the tensor, n_ℓ times the entropy of the distribution over all C_ℓ symbols, zero included, and it does so mainly by moving mass to the zero symbol, under a condition on that distribution which holds in every run we report (Appendix A.6).

The perspective form does more than tie a weight to its mass. The T_1 term is the convex envelope of the function costing $(\bar{w}_i^{(\ell)})^2$ when weight i is kept and nothing when it is dropped, hence the tightest convex relaxation of that on–off decision and not merely one of them (Cacciola et al., 2023). Writing $V_i^{\max}$ for the largest magnitude representable with the sign of weight i, a weight at the representability limit $|\bar{w}_i^{(\ell)}| = V_i^{\max} y_i^{(\ell)}$ costs $T_1 (V_i^{\max})^2 y_i^{(\ell)}$ here against $T_1 (V_i^{\max})^2 (y_i^{(\ell)})^2$ under the linear coupling $|\bar{w}_i^{(\ell)}| \leq V_i^{\max} y_i^{(\ell)}$: the difference vanishes at $y_i^{(\ell)} \in \{0, 1\}$ and is largest halfway between, so the term prices fractional decisions and agrees with the discrete problem at the integers. Taken alone it does nothing else: with $T_2 = T_3 = 0$ the inner minimum sits at $y_i^{(\ell)} = 1$, and Φ_ℓ reduces to an ℓ_2 penalty on the clipped weights, which the ridge-only arm of Section 4.3 finds inert at $T_1 = 10^{-5}$.

Since $c \mapsto c \log_2 c$ increases beyond its minimum at $c = 1/e$, far below any real count, the term prices every non-zero symbol and pushes mass into the excluded zero symbol; at a fixed non-zero mass it acts weakly the other way, because the same sum is minimized at uniform counts (Appendix A.6).

The outer PRESTO objective is $\min_{w, a, \theta_{\text{fp}}} \mathcal{J}$ with

$$\mathcal{J}(w, a, \theta_{\text{fp}}) := \mathcal{L}_{\text{task}}(\widehat{w}(w, a), \theta_{\text{fp}}) + \underbrace{\sum_{\ell \in \Lambda} \Phi_\ell\Big(\bar{w}^{(\ell)}(a_\ell), a_\ell\Big)}_{\mathcal{R}_{\text{PRESTO}}(w, a)}. \tag{8}$$

The task loss contributes its straight-through gradient, and the regularizer contributes a subgradient $\nabla_{\text{PRESTO}} \in \partial_{(w,a)} \mathcal{R}_{\text{PRESTO}} \times \{0\}$ in which the zero appears because $\mathcal{R}_{\text{PRESTO}}$ does not depend on the floating-point parameters, and, up to that approximation, the sum of the two is an element of the generalized gradient of $\mathcal{J}$ when the inner problem is solved to optimality; Section 3 derives its components and states what a truncated dual delivers instead.

[1] With the continuous extension $0 \log_2 0 := 0$.

3 OPTIMIZATION

3.1 LAGRANGIAN RELAXATION AND DECOMPOSITION

The inner problem of Eq. (3) contains one set of variables for every weight and a second set that is shared across the tensor. The representation and mass constraints of Eqs. (4) and (5) involve a single weight index i, whereas each count constraint of Eq. (6) couples all the weights through the sum $c_b^{(\ell)} = \sum_{i \in \mathcal{I}_\ell} x_{i,b}^{(\ell)}$, so that solving the inner problem directly would mean handling every assignment mass of a tensor at once. Lagrangian relaxation removes exactly that global coupling.

Fix a tensor $\ell \in \Lambda$, a clipped latent-weight vector $\bar{w}^{(\ell)}$ and a step size a_ℓ. Attach to each count constraint an unrestricted multiplier $\xi_b^{(\ell)} \in \mathbb{R}$, collected in $\xi^{(\ell)} = (\xi_b^{(\ell)})_{b \in \mathcal{B}}$, with the sign convention $\xi_b^{(\ell)} \big(\sum_{i \in \mathcal{I}_\ell} x_{i,b}^{(\ell)} - c_b^{(\ell)} \big)$. Dualizing only those constraints gives the partial Lagrangian

$$\begin{aligned} \mathscr{L}_\ell(x^{(\ell)}, y^{(\ell)}, c^{(\ell)}; \xi^{(\ell)}) := T_1 \sum_{i \in \mathcal{I}_\ell} \frac{(\bar{w}_i^{(\ell)})^2}{y_i^{(\ell)}} + T_2 \sum_{i \in \mathcal{I}_\ell} y_i^{(\ell)} \\ + T_3 \sum_{b \in \mathcal{B}} c_b^{(\ell)} \log_2 c_b^{(\ell)} + \sum_{b \in \mathcal{B}} \xi_b^{(\ell)} \left(\sum_{i \in \mathcal{I}_\ell} x_{i,b}^{(\ell)} - c_b^{(\ell)} \right), \end{aligned} \tag{9}$$

in which $x^{(\ell)}$ and $y^{(\ell)}$ retain the domain, representation and mass constraints of Eqs. (4)–(7) while the count constraints are no longer imposed, their violation being priced by $\xi^{(\ell)}$. Its minimum over the remaining feasible variables is the dual function $\varphi^{(\ell)}_{\bar{w}^{(\ell)}, a_\ell}(\xi^{(\ell)})$, concave as a pointwise infimum of affine functions of the multipliers, and rearranging Eq. (9) shows that it splits:

$$\begin{aligned} \varphi^{(\ell)}_{\bar{w}^{(\ell)}, a_\ell}(\xi^{(\ell)}) = \sum_{b \in \mathcal{B}} \min_{c_b^{(\ell)} \geq 0} \left\{ T_3 c_b^{(\ell)} \log_2 c_b^{(\ell)} - \xi_b^{(\ell)} c_b^{(\ell)} \right\} \\ + \sum_{i \in \mathcal{I}_\ell} \min_{(x_i^{(\ell)}, y_i^{(\ell)}) \in \mathcal{F}_i^{(\ell)}} \left\{ T_1 \frac{(\bar{w}_i^{(\ell)})^2}{y_i^{(\ell)}} + T_2 y_i^{(\ell)} + \sum_{b \in \mathcal{B}} \xi_b^{(\ell)} x_{i,b}^{(\ell)} \right\}, \end{aligned} \tag{10}$$

where $x_i^{(\ell)} = (x_{i,b}^{(\ell)})_{b \in \mathcal{B}}$, and $\mathcal{F}_i^{(\ell)}$ is the local feasible set

$$\mathcal{F}_i^{(\ell)} := \left\{ (x_i^{(\ell)}, y_i^{(\ell)}) : \begin{array}{l} x_{i,b}^{(\ell)} \geq 0 \quad \text{for every } b \in \mathcal{B}, \\ 0 \leq y_i^{(\ell)} \leq 1, \\ \sum_{b \in \mathcal{B}} v_b^{(\ell)}(a_\ell) x_{i,b}^{(\ell)} = \bar{w}_i^{(\ell)}, \\ \sum_{b \in \mathcal{B}} x_{i,b}^{(\ell)} = y_i^{(\ell)} \end{array} \right\}. \tag{11}$$

The first line consists of B independent one-dimensional count subproblems, one per non-zero symbol, and the second of n_ℓ independent per-weight subproblems, which share the multiplier but no optimization variables. Once $\xi^{(\ell)}$ is fixed the tensor-level problem can therefore be solved in parallel across symbols and across weights, which is the reason for relaxing it. The dual problem is

$$\max_{\xi^{(\ell)} \in \mathbb{R}^B} \varphi^{(\ell)}_{\bar{w}^{(\ell)}, a_\ell}(\xi^{(\ell)}), \tag{12}$$

and it is exact: the feasible set stays compact when the count constraints are perturbed and the objective is closed, so the perturbed value is lower semicontinuous at the origin and no duality gap remains; the maximum is attained as soon as one weight lies strictly inside its representable interval. Proposition 1, in Appendix A.5.1, proves both statements. The hypothesis of the first, that the inner problem is feasible with finite value, is supplied by clipping the latent weights; the attainment of the maximum asks in addition for the strictly interior weight, which clipping does not by itself provide. The relaxation therefore changes the way the inner problem is solved without changing its value. Finally, for a fixed $\xi^{(\ell)}$ the difference between the counts that the per-weight solutions reconstruct and the counts that the count subproblems return is a supergradient of the boxed dual that the implementation ascends (Appendix A.5.2), which coincides with $\varphi^{(\ell)}_{\bar{w}^{(\ell)}, a_\ell}$ where the box is inactive, and that difference drives the dual iteration described below.

3.2 Per-Weight Solution, Gradients and Dual Update

For a fixed multiplier and a fixed non-zero mass y_i, the per-weight subproblem is a linear knapsack over the B non-zero symbols, the zero symbol taking the remaining mass $1 - y_i$, and it is solved exactly, without iteration. Every symbol contributes a point (v_b, ξ_b), and the optimum lies on the lower convex hull of those points, so the non-zero mass concentrates on at most two adjacent vertices of the hull. The optimal non-zero mass y_i^* then follows from a one-dimensional minimization of a perspective term against a linear one, at a cost of $O(C)$ per weight. Appendix A.2.1 states the breakpoints, shows that the vertices of the hull are the only candidates and gives the closed form of y_i^*, and Algorithm 1, in Appendix A.7, is the solver that follows from it.

At a fixed multiplier the two families of subproblems are solved exactly, so Danskin's theorem applies to the inner minimization and the derivative that enters training is an *envelope* derivative evaluated at the inner optimum rather than the derivative of a smoothed surrogate. For the latent weights it is

$$\beta_i^* \;=\; \mu_i \;+\; \frac{2T_1\,\bar{w}_i}{y_i^*}, \tag{13}$$

where μ_i, the slope of the hull between the two symbols to which the weight is assigned, is the multiplier of its representation constraint, and the step sizes receive the matching derivative through the dependence on a_ℓ of the levels v_b of Eq. (2) and of the clipped weights $\bar{w}_i^{(\ell)}(a_\ell)$ of Eq. (15) in Appendix A.1, so that the network, the quantization grid and the symbol distribution all move within one backward pass (Appendices A.3.1–A.3.4).

Three qualifications belong with that statement. We advance the dual by a few warm-started steps per call rather than solving it, so the expression is an exact subgradient of the *relaxed* value function at the multiplier we hold and only an inexact proxy for Φ_ℓ, with the count residual as our diagnostic for that gap. The solver also boxes the relaxed counts, so the problem it poses is not literally Eq. (3). And on an interior vertex of the hull, or on the representability boundary, the multiplier it selects need not be the envelope one. Appendices A.3.1 and A.5.2 state these differences precisely.

The multiplier itself is updated by normalized FISTA-style supergradient ascent with warm starts, using the count residual divided by $c_{\max}$, the parameter count of the network, and every run except the four absolute-step rows of Table 2 measures the step as a fraction of the dual range, the interval to which the multiplier is clipped, because an absolute step makes the entropy term weaker rather than stronger as T_3 grows (Appendices A.4.2 and C.9).

4 Experiments

4.1 Evaluation Protocol

The question we ask throughout is whether PRESTO makes quantized weights smaller after lossless coding than its LSQ baseline, at a comparable accuracy and training budget. Every run starts from a public full-precision checkpoint, quantizes all the convolutional and linear weight tensors and leaves the activations unquantized; unquantized parameters stay in 32-bit floating point (FP32). Optimizers, schedules, distillation and batch-norm recalibration are implementation rather than method (Appendix B.1). The codebook has $C_\ell = 16$ symbols everywhere except on the first two convolutions of AlexNet and on twenty-six early and depthwise tensors of EfficientNet-B0 in Table 1, 0.54% and 3.75% of their weights, which use eight bits and are counted at their true cost. Step sizes are per tensor in every run of Tables 1, 6 and 8; a per-channel variant is used only by superseded runs, made under an earlier recipe: the EfficientNet-B0 rows of Table 2 and all of Tables 9 and 11, priced in Appendix C.7. Each number is the final epoch of a single run unless we say otherwise.

Size accounting. Size is measured rather than nominal. The *packet* is the complete serialized artifact: the support mask, the stream of non-zero symbols, which we call the value stream, and the metadata, learned step sizes included. A nominal bit width would hide both the gain from entropy coding and the cost of the metadata. We count in bits: eight for every byte `zstd` returns on the support mask, packed one bit per weight, and on the value stream, and 32 for each stored step size (Appendix C.7). Like published methods we report over the whole checkpoint: with f the fraction of parameters we quantize, from 99.01% to 99.98% here (Table 4), a packet of r against the quantized

Table 1: Top-1 accuracy and size of the serialized packet, as a percentage of the whole checkpoint, on ImageNet; parentheses give the compression factor and brackets the full-precision accuracy each published method reports. *nominal size, no entropy coding. ‡reconstructs a full-precision network from a latent code at deployment. §a different checkpoint: the Top-1 column gives the drop.

Architecture	Variant	Epochs	Top-1 [%]	Size [%]	Sparsity [%]
ResNet-18 FP32 69.734%	LSQ only	60	70.490	9.84 (10.2×)	18.2
	PRESTO ($10^{-5}, 10^{-7}, 6{\cdot}10^{-8}$), 3 repeats	**60**	**70.020**±.083	**4.89**±.015 (**20.4**×)	61.8
	PRESTO ($10^{-5}, 10^{-7}, 10^{-7}$)	**100**	**69.860**	**4.24** (**23.6**×)	69.7
	HEMP+LOBSTER (Tartaglione et al., 2021)	–	69.70 [69.76]	5.34 (18.7×)	–
	EPR-DFT‡§ (Oktay et al., 2020)	–	0.0 [70.0]	4.22 (23.7×)	–
ResNet-50 FP32 76.122%	LSQ only	20	76.554	9.97 (10.0×)	22.1
	PRESTO ($10^{-5}, 10^{-7}, 3{\cdot}10^{-8}$)	**20**	**76.318**	**5.12** (**19.5**×)	60.0
	HEMP (high acc.) (Tartaglione et al., 2021)	–	74.52 [76.13]	8.88 (11.3×)	–
	NNCodec§ (Becking et al., 2023)	–	−1.01 [80.34]	7.43 (13.5×)	–
DeiT-Small FP32 79.742%	LSQ + distillation	25	79.706	12.00 (8.3×)	13.7
	PRESTO ($10^{-5}, 5{\cdot}10^{-8}, 1.5{\cdot}10^{-8}$), reference	**25**	**79.378**	**9.06** (**11.0**×)	39.7
	PRESTO ($10^{-5}, 10^{-7}, 3{\cdot}10^{-8}$), double	25	78.948	7.74 (12.9×)	48.8
	Q-ViT 3-bit (Li et al., 2022)	300	79.0 [79.9]	9.86* (10.1×)	–
	FTerViT W2A8 (Ruciński et al., 2026)	–	77.47 [79.86]	6.58* (15.2×)	–
ViT-B/16 FP32 81.056%	LSQ + distillation	20	80.816	10.48 (9.5×)	21.8
	PRESTO ($10^{-5}, 5{\cdot}10^{-8}, 1.5{\cdot}10^{-8}$), reference	**20**	**80.372**	**4.42** (**22.6**×)	72.9
	NNCodec (Becking et al., 2023)	–	79.95 [81.07]	9.49 (10.5×)	–
	NNCodec (Becking et al., 2023)	–	75.22 [81.07]	4.85 (20.6×)	–
EfficientNet-B0 FP32 77.566%	LSQ + distillation	30	77.034	12.05 (8.3×)	16.5
	PRESTO ($10^{-5}, 2.1{\cdot}10^{-7}, 6{\cdot}10^{-8}$)	**30**	**76.820**	**10.15** (**9.9**×)	28.0
	NNCodec (Becking et al., 2023)	–	76.78 [77.67]	11.25 (8.9×)	–

weights is $rf + (1 - f)$ of the checkpoint. Sizes are percentages of it, the form in which they add and decompose; Table 1, where published work reports factors, also gives the compression factor, 100 divided by that percentage. Published sizes are converted from the megabytes each method reports, over the uncompressed size it reports, and within a block of Table 1 our rows differ only in (T_1, T_2, T_3) and, where stated, the budget.

Accuracy comparison. Accuracies are compared raw rather than as drops. Every method measures its own full-precision accuracy, and the same checkpoint evaluated by two different codebases does not give the same number: our 79.742% on DeiT-Small is 0.118 accuracy points below the 79.86% that Ruciński et al. (2026) report for DeiT-Small. Comparing drops would credit us with those differences, so we compare top-1 directly wherever the checkpoint is shared and fall back on drops only where the checkpoint differs, the rows marked §. Figure 1 plots drops throughout, which is what puts six networks on one panel.

4.2 Controlled Comparison

Table 1 contains the central measurement, and Figure 1 sets our frontier points against the published points of the same network. Color and marker identify the network: a filled marker on a solid line is a run of ours, an open marker on a dotted line a point of a published NNCodec grid, and a cross a single published point, named beside it where space allows. Appendix C adds further points in Table 6, and Appendix C.8 follows one run epoch by epoch in Figure 3.

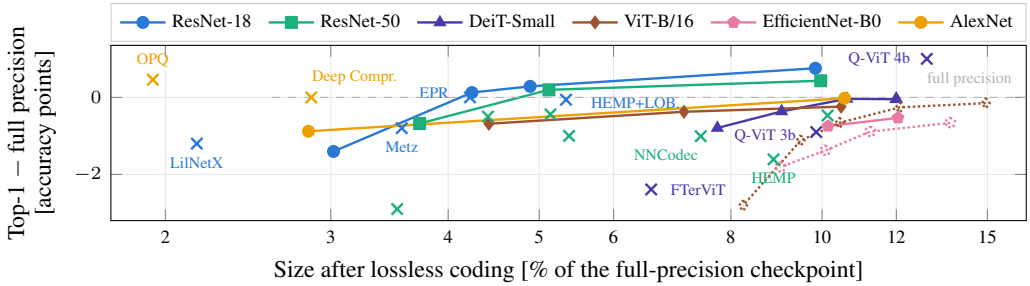

Figure 1: Accuracy–size frontier of our operating points and of the published points, all but one (Appendix C). The vertical axis is each method's drop from its own full precision. Dotted grids are NNCodec's, on ViT-B/16 and EfficientNet-B0; the five unnamed crosses are the published ResNet-50 rows of Table 6.

On ResNet-18, PRESTO halves the packet of the LSQ baseline, from 9.84% of the checkpoint to 4.89%, a compression of $20\times$, and still finishes 0.286 accuracy points above full precision. That row is the mean of three repeats, with standard deviations of 0.083 on the accuracy and 0.015 on the packet, so the margin is 3.4 standard deviations; ResNet-50 reproduces the trade at a third of the budget, roughly halving its LSQ baseline to 5.12% at 0.196 accuracy points above its checkpoint. The LSQ baseline itself finishes 0.756 accuracy points above the checkpoint, so clearing full precision is not a demanding bar here; our claim is the controlled one, that the regularizer roughly halves the size of a quantized ResNet while its accuracy stays above that bar.

Budget, rather than T_3, buys accuracy at almost no cost in size: at $T_3 = 10^{-7}$ the run lands 0.128 accuracy points below the checkpoint at sixty epochs and 0.126 *above* it at one hundred, at 4.22% and 4.24% (Appendix C.3).

On DeiT-Small, ViT-B/16 and EfficientNet-B0 no training recipe of ours reaches full precision, so Table 1 reports the accuracy–size trade-off against the LSQ baseline of each network: PRESTO removes 58% of the ViT-B/16 packet for 0.444 accuracy points, and on all three networks it stays less than 0.75 accuracy points below full precision (Appendix C.5).

4.3 Term Ablation

Five sixty-epoch runs, or arms, on ResNet-18 and four on AlexNet, identical but for the coefficient triple, answer one question: how much of the compression survives if only one term is kept? The ridge alone does nothing, giving 9.87% of the checkpoint against the LSQ baseline's 9.84%, a difference of about two standard deviations of our packet.

Which of the two remaining terms suffices on its own is a property of the architecture rather than of the method. Writing the size reduction of the full objective as 100%, the sparsity term alone recovers 97.5% of it on AlexNet, whereas on ResNet-18 it recovers 58.0% and the entropy term 97.3%.

Both terms produce zeros, the entropy term alone reaching 60.30% sparsity on ResNet-18 against 45.56% for the sparsity term. On that network only the entropy term also shortens the code of the weights that survive, taking that run's own non-zero symbol entropy from 2.784 to 1.926 bits, which we report as a measurement and not as a property of the objective; on AlexNet both terms bring it to about the same value. Nor do the two terms add: the full objective is 2.7% smaller than the better single arm on ResNet-18 and 6.3% smaller on AlexNet, at accuracy differences of 0.070 and 0.126 points, because its non-zero symbol entropy ends where the better single term had already brought it, leaving the other term little to remove (Appendix C.2).

Combining the terms inside one objective also matters. Splitting the same objective on ResNet-18 into two twenty-epoch stages, in either order, reaches the accuracy of the joint run at the same forty-epoch budget, within 0.034 points, but the joint run is 4.2% and 19.3% smaller than the two pipelines, part of which is the re-fit of the grid that a pipeline carries and a joint run does not (Appendix C.4).

The cost lies mainly in the entropy term, which enters the dual solver and multiplies the training time of the LSQ baseline by 3.3 to 12.6, while on ResNet-18 and AlexNet the sparsity term trains at the speed of the LSQ baseline (Appendix B.5). With the shares above this is a practical rule: on a redundant architecture the term that costs nothing buys almost all of the compression, whereas without that redundancy the expensive one separates a packet of 6.97% from one of 4.89%.

4.4 AlexNet

AlexNet is the one network here on which published work is still ahead of us, and the ablation suggests why. 95.96% of its 61.1M weights sit in three fully connected tensors, and redundancy on that scale is what a pruning pipeline exploits, which is why our sparsity term almost suffices here. Both methods ahead of us are prune-first: Deep Compression (Han et al., 2016) reports 2.86% of the checkpoint, the $35\times$ it publishes, at parity with its full-precision accuracy, and OPQ (Hu et al., 2021) reports $+0.46$ at $138.96\times$, a ratio that becomes 1.94% of the checkpoint once the support mask is charged for. Our best point is 2.84%, the same $35\times$, at 0.880 accuracy points below full precision, so at Deep Compression's size we are that much less accurate, and OPQ is smaller than both.

Decomposition of the gap. The gap is the mask rather than the coder. At 79.66% sparsity our packet divides into 1.973% of mask and 0.854% of values, whereas OPQ at 92.30% divides into 1.223% and 0.719%. Per surviving weight our value stream is the more compact, 1.34 bits against 2.99, and the mask is 0.75 of the 0.88 points by which our streams exceed theirs: its i.i.d. cost model $H(p)/32$ falls steeply beyond $p = 0.8$, and we stop thirteen points of sparsity short. Their figure is that cost model applied to their published sparsity and bit width rather than a measurement through our serializer, so it is an estimate. The codebook is not the explanation either, since our grid costs only 0.024 accuracy points here, at 10.57% of the checkpoint.

Source of the deficit. The deficit is neither the budget nor a retraining phase too short to recover from the pruning; we believe it is allocation, since one coefficient drives the sparsity of all eight tensors while both methods ahead of us choose a level per layer. Appendix C.6 gives the measurements and the arithmetic of this hypothesis, which we have not tested in a controlled arm; we have no converged measurement above 80% sparsity here.

4.5 Comparison with Published Work

The ResNets. On ResNet-18 the strongest published point at no loss of accuracy is EPR-DFT (Oktay et al., 2020). Their network is trained from scratch, so only drops compare: they lose none at 4.22% and we gain 0.126 at 4.24%. The sizes differ by about one and a half standard deviations of our packet, so we claim a tie. HEMP+LOBSTER, at 69.70% and 5.34%, is behind us on both axes. On size alone LilNetX is ahead, at 2.16% and 1.2 below its own uncompressed baseline against our deepest point, 3.02% at 1.404: smaller and less degraded. Like EPR it stores a latent code that a learned decoder expands into a full-precision network, where our file is a deployable four-bit model, so the two sizes do not measure the same artifact. On ResNet-50 no published point betters ours on both axes: our 5.12% at $+0.196$ and 3.73% at -0.678 against the seven that Tables 1 and 6 list; the nearest in size is DeepCABAC's sparse point, 5.14% at -0.44.

DeiT-Small. Q-ViT (Li et al., 2022) at three bits reports 79.0% at a nominal 9.86% against our measured 79.378% at 9.06%, so we are more accurate and smaller, although they also quantize the activations. Appendix C.5 discusses the other published points on this network.

ViT-B/16. The standardized codec publishes a full rate–accuracy grid on this network. At 9.49% of the checkpoint NNCodec is already 1.12 points below its full precision, and at 4.85%, the size closest to ours, it gives 75.22% against our 80.372% at 4.42%: five points at a smaller size. The comparison is unusually clean: both are weight-only and both measure the code length on the same checkpoint.

EfficientNet-B0. Here the LSQ baseline lands about half a point below the checkpoint, and NNCodec was ahead of us until two changes of recipe closed the gap: eight bits on the depthwise convolutions and a few early layers, and per-tensor rather than per-channel step sizes, priced in Appendix C.7. PRESTO then gives 76.820% at 10.15% against NNCodec's 76.78% at 11.25%, and at essentially the same size, where their grid gives 76.29% at 10.08%, it is more accurate by 0.53 points. Against that 11.25% row the accuracy margin, 0.04, is inside the run-to-run spread we measure on ResNet-18, while the margin on size, 1.10 points of the checkpoint, is not.

5 Conclusions and Future Work

We presented PRESTO, a regularizer that trains a quantized network against a proxy for its code length: a convex inner problem in perspective form, split by a Lagrangian relaxation, solved exactly at $O(C)$ per weight for a fixed multiplier and differentiated by an envelope argument. On every architecture studied it reduces the serialized packet relative to its LSQ baseline, roughly halving it on both ResNets and more than halving it on ViT-B/16. It has two limits: its entropy term multiplies the training time of the LSQ baseline by 3.3 to 12.6, and on AlexNet prune-first pipelines remain ahead, a deficit that Section 4.4 locates in sparsity allocation. Three directions follow: learning the number of codebook symbols C jointly with the weights, moving to larger models, and extending the objective to the activations.

Reproducibility Statement

The experimental protocol is in Appendix B: the training recipe in Appendix B.1; the configuration of every run, the public checkpoints identified by their content hash, the serialization procedure and the library versions in Appendix B.2; the choices held identical across architectures in Appendix B.3; the procedure by which each coefficient was calibrated in Appendix B.4; and the hardware and the wall-clock cost in Appendix B.5. The conventions under which every size is measured are stated in Section 4.1. The proof of Proposition 1, with its assumptions, and the exact statements for the boxed problem and the truncated dual are in Appendix A.5, and Algorithms 1 and 2 in Appendix A.7 give the inner solver and the training step. One limitation, stated in Appendix B.2, concerns reproduction: the training script sets no global random seed, so the three repeated runs measure run-to-run nondeterminism rather than sensitivity to initialization, and the spread they give is measured on ResNet-18 only.

An anonymized implementation of the regularizer, the dual solver, the training script and the serializer is included in the supplementary material.

Disclosure of AI Use

Required disclosures. Large language models, Claude (Anthropic) through Claude Code and ChatGPT (OpenAI), were used in the following ways. They reviewed the manuscript and gave feedback on the presentation of the results and on the precision of its claims. They occasionally helped diagnose training runs that had failed, while every experiment was designed and chosen by the authors. They helped implement the method in the training code, following the derivations written by the authors, who defined every equation it computes. They also assisted in writing one proof: the model, the regularizer and every mathematical statement in the paper are the authors' own, but the argument by which Proposition 1 excludes a duality gap was rewritten with the assistance of Claude after an AI review pointed out a gap in an earlier version, and the authors checked the new argument and the theorems it invokes against the original source.

Recommended disclosures. They were also used for retrieval and discovery, to search the literature for related work; we checked each reference against its published record. The same tools helped write and debug the rest of the training code, edit the manuscript for readability and draft parts of its text. No generative AI tool was used to generate data or to produce any reported measurement: every number in the paper comes from training runs executed and logged by the authors.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

D. Becking, P. Haase, H. Kirchhoffer, K. Müller, W. Samek, and D. Marpe. NNCodec: An open source software implementation of the neural network coding ISO/IEC standard. In *ICML Workshop on Neural Compression: From Information Theory to Applications*, 2023.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

J. F. Bonnans and A. Shapiro. *Perturbation Analysis of Optimization Problems*. Springer, 2000.

M. Cacciola, A. Frangioni, X. Li, and A. Lodi. Deep neural networks pruning via the structured perspective regularization. *SIAM Journal on Mathematics of Data Science*, 5(4):1051–1077, 2023.

S. Ceria and J. Soares. Convex programming for disjunctive convex optimization. *Mathematical Programming*, 86(3):595–614, 1999.

J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan. PACT: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

A. Conzelmann and R. Bamler. Reducing storage of pretrained neural networks by rate-constrained quantization and entropy coding. *arXiv preprint arXiv:2505.18758*, 2025.

J. M. Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966.

Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *IEEE/CVF International Conference on Computer Vision*, 2019.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020.

A. Frangioni and C. Gentile. Perspective cuts for a class of convex 0–1 mixed integer programs. *Mathematical Programming*, 106(2):225–236, 2006.

S. Girish, K. Gupta, S. Singh, and A. Shrivastava. LilNetX: Lightweight networks with extreme model compression and structured sparsification. In *International Conference on Learning Representations*, 2023.

O. Günlük and J. Linderoth. Perspective reformulations of mixed integer nonlinear programs with indicator variables. *Mathematical Programming*, 124(1–2):183–205, 2010.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations*, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

P. Hu, X. Peng, H. Zhu, M. M. Sabry Aly, and J. Lin. OPQ: Compressing deep neural networks with one-shot pruning-quantization. In *AAAI Conference on Artificial Intelligence*, pp. 7780–7788, 2021.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2021.

Y. Li, S. Xu, B. Zhang, X. Cao, P. Gao, and G. Guo. Q-ViT: Accurate and fully quantized low-bit vision transformer. In *Advances in Neural Information Processing Systems*, 2022.

C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations*, 2018.

C. Metz, O. Bichler, and A. Dupret. Efficient neural compression with inference-time decoding. In *IEEE International Symposium on Circuits and Systems*, 2024.

D. Oktay, J. Ballé, S. Singh, and A. Shrivastava. Scalable model compression by entropy penalized reparameterization. In *International Conference on Learning Representations*, 2020.

R. T. Rockafellar. *Conjugate Duality and Optimization*. CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, Philadelphia, 1974.

S. Ruciński, P. Bonazzi, E. Türetken, S. Narduzzi, M. Magno, and N. Maamari. FTerViT: Fully ternary vision transformer. *arXiv preprint arXiv:2605.21171*, 2026.

Y. Shi, S. Bai, X. Wei, R. Gong, and J. Yang. Lossy and lossless (L^2) post-training model size compression. In *IEEE/CVF International Conference on Computer Vision*, pp. 17500–17510, 2023.

M. Tan and Q. V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 2019.

E. Tartaglione, S. Lathuilière, A. Fiandrotti, M. Cagnazzo, and M. Grangetto. HEMP: High-order entropy minimization for neural network compression. *Neurocomputing*, 461:244–253, 2021.

E. Tartaglione, A. Bragagnolo, A. Fiandrotti, and M. Grangetto. LOss-Based SensiTivity rEgulaRization: Towards deep sparse neural networks. *Neural Networks*, 146:230–237, 2022.

H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2021.

K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations*, 2017.

J. Wenshøj, T. Chen, B. Pepin, and R. Selvan. CoDeQ: End-to-end joint model compression with dead-zone quantizer for high-sparsity and low-precision networks. *arXiv preprint arXiv:2512.12981*, 2025.

S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinc, D. Neumann, A. Osman, D. Marpe, H. Schwarz, T. Wiegand, and W. Samek. DeepCABAC: Context-adaptive binary arithmetic coding for deep neural network compression. In *ICML Joint Workshop on On-Device Machine Learning and Compact Deep Neural Network Representations*, 2019a.

S. Wiedemann, A. Marban, K.-R. Müller, and W. Samek. Entropy-constrained training of deep neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2019b.

S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinc, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand, D. Marpe, and W. Samek. DeepCABAC: A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):700–714, 2020.

A SUPPLEMENTARY DERIVATIONS

The perspective form of a separable convex term controlled by an indicator, and the fact that it is the tightest convex relaxation of that indicator, are due to Ceria and Soares (1999), Frangioni and Gentile (2006) and Günlük and Linderoth (2010); Cacciola et al. (2023) carried it to structured pruning, and Section 2.2 carries it to the assignment of a weight across codebook symbols. Being the tightest such relaxation is what separates it from the stochastic gates of ℓ_0 regularization (Louizos et al., 2018), which relax the same keep-or-drop decision with a smooth surrogate instead.

The pieces of Section 2 are not independent, and the order in which they constrain one another is short enough to state. Learning the step size, which is what LSQ does, turns a_ℓ from a constant fitted before training into a variable that moves with the weights. The codebook $v_b^{(\ell)} = a_\ell q_b$ of Eq. (2) moves with it, so the levels on which the inner problem represents each weight are themselves being optimized. The regularizer therefore acquires a dependence on a_ℓ that a fixed grid would not have. That dependence has two channels, the levels of Eq. (4) and the clipped weight $\bar{w}^{(\ell)}(a_\ell)$ that the inner problem takes as its argument, and both have to be differentiated if the step sizes are to be trained against the code length rather than against the task loss alone. Clipping is what makes the second channel exist, and it does one more thing: the representation constraint of Eq. (4) admits a solution only when $|\bar{w}_i^{(\ell)}|$ does not exceed $V_i^{\max}$, which is exactly what clipping guarantees, so the inner problem is feasible by construction rather than by assumption. Feasibility is in turn the hypothesis under which the relaxation of Section 3.1 is exact. The derivations that follow are the consequences of this chain, and this appendix is ordered along it.

A.1 LEARNED-STEP QUANTIZATION: FULL STATEMENT

Let $Q_n < 0 < Q_p$ be two integers denoting the minimum and maximum integer codes, respectively. The integer codebook is

$$\mathcal{Q} := \{q \in \mathbb{Z} : Q_n \le q \le Q_p\}, \qquad C := |\mathcal{Q}| = Q_p - Q_n + 1,$$

where $\mathbb{Z}$ is the set of integers and C is the number of symbols. We associate each tensor ℓ with a quantization step size $a_\ell \in \mathbb{R}_{>0}$. The collection of step sizes is $a = \{a_\ell\}_{\ell \in \Lambda}$.

For $\alpha, \beta, u \in \mathbb{R}$ with $\alpha \le \beta$, define

$$\mathrm{clip}(u; \alpha, \beta) := \min\{\beta, \max\{\alpha, u\}\}.$$

Moreover, let $\mathrm{round}(u)$ denote the integer nearest to u, with ties resolved toward the even integer. For weight $w_i^{(\ell)}$, we set

$$r_i^{(\ell)} := \frac{w_i^{(\ell)}}{a_\ell}, \tag{14}$$

$$\bar{w}_i^{(\ell)}(a_\ell) := a_\ell \ \mathrm{clip}\Big(r_i^{(\ell)}; Q_n, Q_p\Big), \tag{15}$$

$$\widehat{w}_i^{(\ell)} := a_\ell \ \mathrm{round}\left(\frac{\bar{w}_i^{(\ell)}(a_\ell)}{a_\ell}\right). \tag{16}$$

$r_i^{(\ell)}$ is the weight normalized by the step size. The clipped latent weight $\bar{w}_i^{(\ell)}(a_\ell)$ remains real-valued but is restricted to the representable interval $[a_\ell Q_n, a_\ell Q_p]$. Rounding its normalized value then produces $\widehat{w}_i^{(\ell)}$, which belongs to the real-valued codebook $a_\ell \mathcal{Q}$ and is used in the forward pass. The quantized weights of the entire network are $\widehat{w}(w, a) = \{\widehat{w}^{(\ell)}\}_{\ell \in \Lambda}$.

PRESTO uses Learned Step Size Quantization (LSQ) (Esser et al., 2020) to propagate a surrogate gradient through the rounding operation. The PRESTO inner problem operates on $\bar{w}_i^{(\ell)}$, ensuring that the quantizer and the regularizer share the same representable domain.

A.2 SOLVING THE INNER PROBLEM EXACTLY

At a fixed multiplier the decomposition of Section 3.1 leaves two families of subproblems, one per weight and one per symbol. Both are solved exactly and without iteration, and this section gives each in turn.

A.2.1 SOLVING THE PER-WEIGHT SUBPROBLEM

Fix a tensor $\ell \in \Lambda$. Throughout this section, we temporarily omit the superscript (ℓ), with the understanding that all quantities refer to the same tensor. Accordingly, $\xi = (\xi_b)_{b\in\mathcal{B}}$ denotes the multiplier vector introduced in Section 3.1 with its tensor superscript omitted.

For fixed ξ, the problem separates over the weight index i. Given a weight $\bar{w}_i$ and a fixed mass $y_i \in [0, 1]$, define

$$K_i(y_i; \bar{w}_i, \xi, v) := \min_{x_i} \quad \sum_{b\in\mathcal{B}} \xi_b x_{i,b} \tag{17}$$

$$\text{s.t.} \quad \begin{cases} \sum_{b\in\mathcal{B}} v_b x_{i,b} = \bar{w}_i, & \sum_{b\in\mathcal{B}} x_{i,b} = y_i, \\ x_{i,b} \geq 0, & b \in \mathcal{B}. \end{cases} \tag{18}$$

The letter K denotes the optimal value of this linear knapsack-type subproblem, and the subscript i identifies the corresponding weight. Here, $x_i = (x_{i,b})_{b\in\mathcal{B}}$ is the vector of assignment masses for weight i, and $v = (v_b)_{b\in\mathcal{B}}$ is the vector of real-valued levels.

For $y_i > 0$, let

$$u_{i,b} := \frac{x_{i,b}}{y_i}, \qquad \widetilde{w}_i := \frac{\bar{w}_i}{y_i}.$$

The vector $u_i = (u_{i,b})_{b\in\mathcal{B}}$ has nonnegative entries summing to one, while $\widetilde{w}_i$ is the rescaled target. It follows that

$$K_i(y_i; \bar{w}_i, \xi, v) = y_i\, K_i(1; \widetilde{w}_i, \xi, v). \tag{19}$$

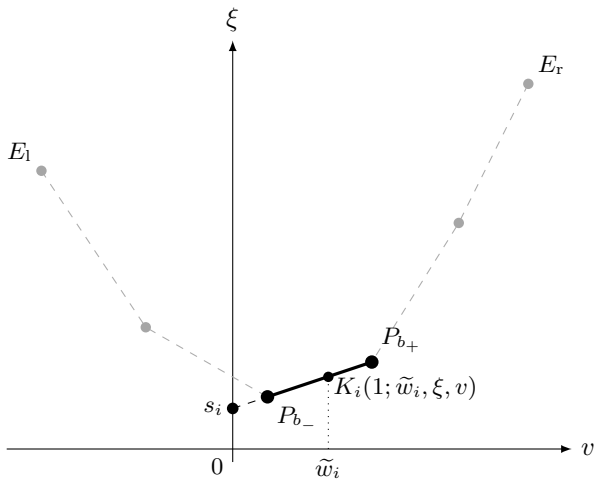

Figure 2: Geometric solution of the normalized per-weight subproblem.

To evaluate $K_i(1; \widetilde{w}_i, \xi, v)$, we associate each symbol b with the point $P_b := (v_b, \xi_b) \in \mathbb{R}^2$. The optimal value is the ordinate of the lower convex hull of the points P_b at the abscissa $v = \widetilde{w}_i$. Let E_{l} and E_{r} denote, respectively, the leftmost and rightmost vertices of the lower hull, as shown in Figure 2. Suppose first that $v_{b_-} < \widetilde{w}_i < v_{b_+}$, where P_{b_-} and P_{b_+} are the endpoints of the lower-hull facet containing this abscissa. We call these points active because they determine the facet and are the only symbols assigned positive mass in the corresponding two-point optimal representation. If $\widetilde{w}_i$ coincides with a lower-hull vertex v_b, the optimal representation assigns unit mass to symbol b. If P_b is an interior vertex of the lower hull, either of its two incident facets may be used in the formulas below. If P_b is one of the two extreme vertices E_{l} and E_{r}, only one incident facet exists. In both cases, the formulas yield unit mass for symbol b and zero mass for the other vertex of the selected facet.

Consider the two-point optimal representation associated with the selected facet. All masses outside its endpoints are set to zero, while the endpoint masses satisfy

$$u_{i,b_-} + u_{i,b_+} = 1, \qquad v_{b_-} u_{i,b_-} + v_{b_+} u_{i,b_+} = \widetilde{w}_i.$$

Solving this two-equation linear system yields

$$u_{i,b_-} = \frac{v_{b_+} - \widetilde{w}_i}{v_{b_+} - v_{b_-}}, \qquad u_{i,b_+} = \frac{\widetilde{w}_i - v_{b_-}}{v_{b_+} - v_{b_-}}. \tag{20}$$

Both masses are strictly positive when $\widetilde{w}_i$ lies in the interior of the facet; if it coincides with an endpoint, the corresponding masses are 1 and 0.

The line containing the active facet can be written as $\xi = \mu_i v + s_i$. Requiring this line to pass through P_{b_-} and P_{b_+} yields

$$\mu_i := \frac{\xi_{b_+} - \xi_{b_-}}{v_{b_+} - v_{b_-}}, \qquad s_i := \frac{\xi_{b_-} v_{b_+} - \xi_{b_+} v_{b_-}}{v_{b_+} - v_{b_-}}. \tag{21}$$

Therefore,

$$K_i(1; \widetilde{w}_i, \xi, v) = \mu_i \widetilde{w}_i + s_i. \tag{22}$$

To characterize the feasible interval of y_i, define

$$V_i^{\max} := \max\{|v_b| : b \in \mathcal{B},\ \bar{w}_i v_b \geq 0\}.$$

If $\bar{w}_i = 0$, define $y_i^{\min} := 0$; otherwise, define

$$y_i^{\min} := \frac{|\bar{w}_i|}{V_i^{\max}}.$$

$V_i^{\max}$ is the largest representable magnitude having the same sign as $\bar{w}_i$, whereas $y_i^{\min}$ is the smallest mass that can represent $\bar{w}_i$.

It remains to minimize the scalar function

$$G_i(y_i) := K_i(y_i; \bar{w}_i, \xi, v) + T_1 \frac{\bar{w}_i^2}{y_i} + T_2 y_i, \qquad y_i \in [y_i^{\min}, 1], \tag{23}$$

where G_i is the value of the subproblem associated with weight i. On an interval over which the active facet remains unchanged, $K_i(y_i; \bar{w}_i, \xi, v) = \mu_i \bar{w}_i + s_i y_i$. If $s_i + T_2 > 0$, the unique stationary candidate in the interior of that interval is

$$y_i^{\text{cand}} := |\bar{w}_i| \sqrt{\frac{T_1}{s_i + T_2}}. \tag{24}$$

Here s_i is read from the facet active on each interval, through Eq. (21); every candidate lying in its interval is compared with the interval endpoints and with those of $[y_i^{\min}, 1]$, the best is y_i^*, and the zero symbol takes $z_i^* := 1 - y_i^*$.

Breakpoints of the active facet. The facet that supports $\widetilde{w}_i = \bar{w}_i / y_i$ changes only when $\widetilde{w}_i$ crosses a vertex of the lower hull. Let $v_{(1)}, \ldots, v_{(m)}$ be the abscissas of the hull vertices that have the same sign as $\bar{w}_i$, sorted by increasing magnitude. As y_i decreases from 1 to $y_i^{\min}$, the ratio $|\widetilde{w}_i| = |\bar{w}_i| / y_i$ increases monotonically from $|\bar{w}_i|$ to $V_i^{\max}$. The active facet therefore changes exactly at

$$y_{(k)} := \frac{|\bar{w}_i|}{|v_{(k)}|}, \qquad y_{(k)} \in [y_i^{\min}, 1], \tag{25}$$

and these values, together with $y_i^{\min}$ and 1, partition the feasible interval into at most $m + 1 \leq C$ subintervals. The pair (μ_i, s_i) is constant on each subinterval, so Eq. (24) applies there unchanged. The breakpoints require no search: they are obtained in closed form from the hull vertices, which are shared by every weight of the tensor.

Computational cost. The lower hull of the $B = C - 1$ points (v_b, ξ_b) does not depend on i. It is built once per tensor and per dual iteration by a left-to-right scan of the sorted codebook values, which finds each next vertex as the point of least slope from the current one, in at most $O(C^2)$ operations. For a single weight, the sweep visits every facet of the hull, at most $B - 1$, and performs constant work in each, namely one evaluation of Eq. (24), the evaluation of G_i at that candidate, at the two breakpoints of the facet and at the endpoints of $[y_i^{\min}, 1]$, and one test that $\bar{w}_i / y_i$ lies on the facet. The per-weight cost is therefore $O(C)$. One pass over a tensor with n_ℓ weights costs $O(C^2 + n_\ell C)$, and a dual update with N_{dual} iterations costs N_{dual} such passes. Since C is a small constant (16 on every tensor except the eight-bit ones of Section 4.1, where it is 256) and the sweep is vectorized over weights, the measured cost of the solver is a small multiple of one elementwise pass over the tensor; Appendix B.5 reports the wall-clock effect.

The zero weight. One case has to be singled out. If $\bar{w}_i = 0$ then $y_i^{\min} = 0$, the whole feasible interval is available and, whenever the active prices are non-negative, the minimizer is $y_i^* = 0$ with $x_i = 0$, so the expression $\mu_i + 2T_1\bar{w}_i/y_i^*$ of Eq. (13) is $0/0$ and does not define a value. It is a genuine kink rather than an artefact: along the stationary branch $y_i = |\bar{w}_i|\sqrt{T_1/(s_i + T_2)}$ the perspective part tends to $\pm 2\sqrt{T_1(s_i + T_2)}$ as $\bar{w}_i \to 0^{\pm}$, so the two one-sided limits bracket the origin and the subdifferential of G_i^* at $\bar{w}_i = 0$ is an interval containing zero when the active prices are non-negative; the implementation selects $\beta_i^* = 0$ in every case, which concerns only latent weights that are exactly zero, and Algorithm 1 states the branch explicitly.

A.2.2 Count Solutions and Dual Residuals

Fix a tensor $\ell \in \Lambda$ and a multiplier vector $\xi^{(\ell)}$. For each symbol $b \in \mathcal{B}$, the solutions of the per-weight subproblems determine the reconstructed count

$$\widehat{c}_b^{(\ell)} := \sum_{i \in \mathcal{I}_\ell} x_{i,b}^{(\ell)*}. \tag{26}$$

The hat distinguishes this quantity, which is reconstructed by summing the optimal assignment masses, from the independent variable $c_b^{(\ell)}$ in the count subproblem.

For $T_3 > 0$, the count subproblem appearing in the first line of Eq. (10) is

$$\min_{c_b^{(\ell)} \geq 0} \left\{ T_3 c_b^{(\ell)} \log_2 c_b^{(\ell)} - \xi_b^{(\ell)} c_b^{(\ell)} \right\}.$$

Its objective is strictly convex for $c_b^{(\ell)} > 0$ and its derivative on this interval is

$$\frac{T_3}{\ln(2)} \left(\ln c_b^{(\ell)} + 1 \right) - \xi_b^{(\ell)}.$$

Setting this derivative to zero and solving for $c_b^{(\ell)}$ gives the unconstrained stationary value

$$c_{b,\mathrm{unc}}^{(\ell)} = \exp\left(\frac{\ln(2)\, \xi_b^{(\ell)}}{T_3} - 1 \right). \tag{27}$$

For numerical stability, the implementation restricts the count to $[c_{\min}, c_{\max}]$, where $c_{\min} > 0$ is a fixed numerical lower bound and $c_{\max}$ is an upper clamp that no count can exceed, since a count is at most the number of weights that can carry it. In the runs of this paper $c_{\max} = N := |\theta|$, the number of parameters of the whole network, quantized and floating-point alike, so it is one value per run rather than one per tensor; the upper clamp binds only at the first dual call, where the initialization places the multipliers on the upper bound (Appendix B.2). The value actually used by the dual solver is therefore

$$c_b^{(\ell)*} := \mathrm{clip}\left(c_{b,\mathrm{unc}}^{(\ell)}; c_{\min}, c_{\max} \right). \tag{28}$$

When $T_3 = 0$, the entropic count component is disabled and Eqs. (27) and (28) are not used.

A recommendation for reproduction. The choice $c_{\max} = N$ is looser than it needs to be, and we record here what we would do differently. A count cannot exceed the number of weights that can carry it, so the tight bound is n_ℓ, and it is also the better-conditioned one: the residual of Eq. (44) is divided by $c_{\max}$ while the step is a fixed fraction of the dual interval, so with a global $c_{\max}$ the dual of a tensor advances N/n_ℓ times more slowly than it would under the tight bound. On ResNet-18 the smallest quantized tensor holds $8{,}192$ weights against the network's 11.7M, so its dual advances some $1{,}400$ times more slowly. Every number in this paper was produced with $c_{\max} = N$, and we have not rerun the experiments with the tight bound, so we do not know how much it would change them; the released implementation exposes both through `--c_max_scope` and defaults to `network`, which reproduces our runs, while we recommend `tensor` for new work.

The two count values in Eqs. (26) and (28) solve different subproblems and need not agree for an arbitrary multiplier vector. Their difference

$$s_b^{(\ell)} := \widehat{c}_b^{(\ell)} - c_b^{(\ell)*} \tag{29}$$

is precisely the residual of the relaxed count constraint. It is also a supergradient of the concave dual function with respect to $\xi_b^{(\ell)}$. More precisely it is a supergradient of the *boxed* dual $\varphi_\ell^{\square}$ and not of $\varphi^{(\ell)}_{\bar{w}^{(\ell)},a_\ell}$, because the count that enters it is the clipped $c_b^{(\ell)*}$ of Eq. (28) rather than the unconstrained stationary value[2]. Thus, $s_b^{(\ell)} = 0$ means that the relaxed constraint is satisfied for symbol b. If a subproblem has multiple minimizers, any consistent selection yields a valid supergradient.

A.3 The Training Direction

The quantities derived here are the ones the optimizers receive. The first three subsections give the pieces, one for the latent weights, one for the multiplier that carries the representation constraint and one for the step sizes; the fourth assembles them together with the task-loss gradient, and the fifth fixes the order in which they must be evaluated.

A.3.1 Regularizer Subgradient with Respect to the Weights

Restoring the tensor superscript ℓ, let $\mu_i^{(\ell)}$ be an optimal Lagrange multiplier associated with the representation constraint of Eq. (4) for weight i, under the sign convention

$$\bar{w}_i^{(\ell)} - \sum_{b\in\mathcal{B}} a_\ell q_b x_{i,b}^{(\ell)} = 0.$$

The full inner Lagrangian, obtained by adding the representation constraints to the partial Lagrangian in Eq. (9), is

$$\mathscr{L}_\ell^{\text{full}} := \mathscr{L}_\ell + \sum_{i\in\mathcal{I}_\ell} \mu_i^{(\ell)} \left(\bar{w}_i^{(\ell)} - \sum_{b\in\mathcal{B}} a_\ell q_b x_{i,b}^{(\ell)} \right).$$

For a convex parametric optimization problem the optimal Lagrange multipliers describe how the optimal-value function responds to perturbations of the constraints, and when the parameter also appears explicitly in the objective its direct contribution has to be added to the contribution of the multiplier (Bonnans and Shapiro, 2000).

The connection with the per-weight function G_i in Eq. (23) can be seen directly. Temporarily omitting the tensor superscript, Eqs. (19) and (22) give, on the selected facet,

$$K_i(y_i; \bar{w}_i, \xi, v) = \mu_i \bar{w}_i + s_i y_i.$$

Consequently,

$$\mu_i \in \partial_{\bar{w}_i} K_i(y_i; \bar{w}_i, \xi, v),$$

where the inclusion accounts for a possible lower-hull breakpoint. From Eq. (23) we obtain, for fixed y_i,

$$\mu_i + 2T_1 \frac{\bar{w}_i}{y_i} \in \partial_{\bar{w}_i} G_i(y_i).$$

To make the minimization over y_i explicit, define the optimized per-weight value

$$G_i^*(\bar{w}_i; \xi, v) := \min_{y_i\in[y_i^{\min},1]} G_i(y_i; \bar{w}_i, \xi, v), \qquad y_i^* \in \arg\min_{y_i\in[y_i^{\min},1]} G_i(y_i; \bar{w}_i, \xi, v).$$

In this notation G_i^* denotes the optimal value while y_i^* denotes a selected minimizer. The dependence of G_i on $\bar{w}_i$, ξ, and v was suppressed in Eq. (23). At a differentiability point where y_i^* is an interior minimizer, formally applying the chain rule to $G_i^*(\bar{w}_i; \xi, v) = G_i(y_i^*(\bar{w}_i); \bar{w}_i, \xi, v)$ gives

$$\frac{dG_i^*}{d\bar{w}_i} = \frac{\partial G_i}{\partial \bar{w}_i} + \frac{\partial G_i}{\partial y_i}\frac{dy_i^*}{d\bar{w}_i} = \frac{\partial G_i}{\partial \bar{w}_i},$$

because $\left.\frac{\partial G_i}{\partial y_i}\right|_{y_i=y_i^*} = 0$ at the interior optimum. Danskin's theorem and the sensitivity theory of convex value functions extend this conclusion to lower-hull breakpoints and boundary minimizers:

[2]The coefficient of $\xi_b^{(\ell)}$ in the partial Lagrangian is $\sum_i x_{i,b}^{(\ell)} - c_b^{(\ell)}$, and evaluating this coefficient at the minimizers selected under the box gives Eq. (29); the two duals coincide where the box is inactive.

a compatible subgradient is obtained from an optimal primal–dual solution while holding y_i^* fixed, provided the multiplier μ_i is optimal for the minimization in y_i as well (Danskin, 1966; Bonnans and Shapiro, 2000). Therefore, no derivative of y_i^* is required, and

$$\mu_i + 2T_1 \frac{\bar{w}_i}{y_i^*} \in \partial_{\bar{w}_i} G_i^*(\bar{w}_i; \xi, v).$$

Restoring the tensor superscript, and observing that the count subproblems contain no explicit dependence on $\bar{w}_i^{(\ell)}$, yields

$$\beta_i^{(\ell)*} := \mu_i^{(\ell)} + 2T_1 \frac{\bar{w}_i^{(\ell)}}{y_i^{(\ell)*}} \in \partial_{\bar{w}_i^{(\ell)}} \Phi_\ell\Big(\bar{w}^{(\ell)}, a_\ell\Big)\Big|_{a_\ell}. \tag{30}$$

Thus, $\mu_i^{(\ell)}$ is the contribution induced by the representation constraint, whereas the second term is the direct contribution of the perspective penalty. Everything in this subsection and the next is written at a dual optimum, where $\varphi^{(\ell)}$ and Φ_ℓ coincide; the implementation evaluates the same expressions at the multiplier it currently holds, and Appendix A.5.2 states what they are exact for in that case. If Φ_ℓ is differentiable with respect to $\bar{w}_i^{(\ell)}$, the subgradient is unique and

$$\beta_i^{(\ell)*} = \frac{\partial \Phi_\ell}{\partial \bar{w}_i^{(\ell)}}\bigg|_{a_\ell}.$$

At a weight for which $\bar{w}_i^{(\ell)}/y_i^{(\ell)*}$ coincides with a hull vertex, Φ_ℓ may be nondifferentiable and the multiplier is fixed by stationarity in $y_i^{(\ell)}$, as the next paragraph and the one after Eq. (31) state; elsewhere the facet, and hence $\mu_i^{(\ell)}$, is unique. If the optimal normalized value reaches the extreme level $v_e^{(\ell)}$, namely $\bar{w}_i^{(\ell)}/y_i^{(\ell)*} = v_e^{(\ell)}$, representability forces all non-zero mass onto the corresponding extreme symbol $e \in \mathcal{B}$. The Karush–Kuhn–Tucker (KKT) conditions then yield

$$\mu_i^{(\ell)} = \frac{\xi_e^{(\ell)} + T_2}{v_e^{(\ell)}} - T_1 v_e^{(\ell)} \tag{31}$$

when $y_i^{(\ell)*} = y_i^{(\ell),\min} < 1$. If the weight itself coincides with the extreme level, so that $y_i^{(\ell)*} = 1$, the upper bound $y_i^{(\ell)} \le 1$ is active as well and its multiplier need not vanish; Eq. (31) sets that multiplier to zero, which selects one multiplier of the representation constraint but in general not a subgradient of G_i^* there. Here, $\xi_e^{(\ell)}$ is the dual multiplier associated with the count-defining constraint for symbol e in Eq. (6). Appendix A.3.2 derives Eq. (31) and explains the endpoint selection.

Interior vertices in the implementation. The derivation of Eq. (31) uses only that all non-zero mass sits on one symbol and that $y_i^{(\ell)*} < 1$, so it applies unchanged when $\bar{w}_i^{(\ell)}/y_i^{(\ell)*}$ coincides with an interior vertex $v_b^{(\ell)}$ of the lower hull, with b in place of e; on an interval of weights around such a point G_i^* is linear with slope $(\xi_b^{(\ell)} + T_2)/v_b^{(\ell)} + T_1 v_b^{(\ell)}$. The implementation applies Eq. (31) only at the extreme symbol. At an interior vertex it takes the slope of one of the two adjacent facets, which is a subgradient of G_i at the fixed $y_i^{(\ell)*}$ but in general not of G_i^*, so on those weights $\beta_i^{(\ell)*}$ and the step-size derivative built from $\mu_i^{(\ell)}$ are not the envelope derivatives of this appendix. Every result in this paper was produced by that implementation, which the released code reproduces. In short runs on synthetic data such weights were between 5% and 17% of a tensor at a dual call; we did not measure their share in the reported runs.

A.3.2 KKT Multiplier at a Representability Boundary

Here we derive the representation multiplier reported in Eq. (31). Fix a tensor ℓ and a weight $i \in \mathcal{I}_\ell$, and suppose that its optimal mass attains the representability lower bound

$$y_i^{(\ell)*} = y_i^{(\ell),\min} = \frac{|\bar{w}_i^{(\ell)}|}{|v_e^{(\ell)}|},$$

where $v_e^{(\ell)}$ is the extreme codebook level having the same sign as $\bar{w}_i^{(\ell)}$. Equivalently,

$$\frac{\bar{w}_i^{(\ell)}}{y_i^{(\ell)*}} = v_e^{(\ell)}.$$

Because no codebook level extends beyond $v_e^{(\ell)}$ in that direction, the representation and mass constraints can hold only if all non-zero mass is assigned to symbol e:

$$x_{i,e}^{(\ell)*} = y_i^{(\ell)*}, \qquad x_{i,b}^{(\ell)*} = 0 \quad \text{for every } b \neq e.$$

Let $\rho_i^{(\ell)} \in \mathbb{R}$ be the Lagrange multiplier associated with the mass constraint

$$\sum_{b\in\mathcal{B}} x_{i,b}^{(\ell)} - y_i^{(\ell)} = 0.$$

For every $b \in \mathcal{B}$, let $\gamma_{i,b}^{(\ell)} \geq 0$ be the multiplier associated with the nonnegativity constraint written as $-x_{i,b}^{(\ell)} \leq 0$. Using for the representation constraint the sign convention introduced in Appendix A.3.1, define the per-weight KKT Lagrangian relevant to the derivation as

$$\begin{aligned}\mathscr{L}_{i,\ell}^{\mathrm{KKT}} := T_1 \frac{(\bar{w}_i^{(\ell)})^2}{y_i^{(\ell)}} + T_2 y_i^{(\ell)} + \sum_{b\in\mathcal{B}} \xi_b^{(\ell)} x_{i,b}^{(\ell)} \\ + \mu_i^{(\ell)} \left(\bar{w}_i^{(\ell)} - \sum_{b\in\mathcal{B}} v_b^{(\ell)} x_{i,b}^{(\ell)} \right) + \rho_i^{(\ell)} \left(\sum_{b\in\mathcal{B}} x_{i,b}^{(\ell)} - y_i^{(\ell)} \right) - \sum_{b\in\mathcal{B}} \gamma_{i,b}^{(\ell)} x_{i,b}^{(\ell)}.\end{aligned}$$

Complementary slackness requires $\gamma_{i,e}^{(\ell)} x_{i,e}^{(\ell)*} = 0$; since $x_{i,e}^{(\ell)*} > 0$, it follows that $\gamma_{i,e}^{(\ell)} = 0$.[3] Stationarity with respect to the active mass, evaluated at the selected optimum, therefore gives

$$\left.\frac{\partial \mathscr{L}_{i,\ell}^{\mathrm{KKT}}}{\partial x_{i,e}^{(\ell)}}\right|_{\substack{x_i^{(\ell)}=x_i^{(\ell)*} \\ y_i^{(\ell)}=y_i^{(\ell)*}}} = \xi_e^{(\ell)} - \mu_i^{(\ell)} v_e^{(\ell)} + \rho_i^{(\ell)} - \gamma_{i,e}^{(\ell)} = \xi_e^{(\ell)} - \mu_i^{(\ell)} v_e^{(\ell)} + \rho_i^{(\ell)} = 0. \tag{32}$$

When $y_i^{(\ell)*} < 1$, the upper bound $y_i^{(\ell)} \leq 1$ is inactive. Stationarity with respect to $y_i^{(\ell)}$ then yields

$$-T_1 \frac{(\bar{w}_i^{(\ell)})^2}{(y_i^{(\ell)*})^2} + T_2 - \rho_i^{(\ell)} = 0. \tag{33}$$

Using $\bar{w}_i^{(\ell)}/y_i^{(\ell)*} = v_e^{(\ell)}$ in Eq. (33) gives

$$\rho_i^{(\ell)} = T_2 - T_1 (v_e^{(\ell)})^2.$$

Substitution into Eq. (32) produces

$$\mu_i^{(\ell)} = \frac{\xi_e^{(\ell)} + T_2}{v_e^{(\ell)}} - T_1 v_e^{(\ell)},$$

which is Eq. (31).

The endpoint case remains to be clarified. If $\bar{w}_i^{(\ell)} = v_e^{(\ell)}$, then representability forces $y_i^{(\ell)*} = 1$ and the upper-bound multiplier for $y_i^{(\ell)} \leq 1$ may be non-zero, so that the representation multiplier is not necessarily unique, and in that situation Equation (31) specifies the multiplier obtained by setting that upper-bound multiplier to zero. This is a selection and not a limit. Where the optimal mass reaches one before the weight reaches the extreme level, the representation multiplier at interior weights is the slope of a hull facet and tends to a different value, so at $|\bar{w}_i^{(\ell)}| = V_i^{\max}$ the quantity the implementation returns is in general not a subgradient of G_i^*, as for the interior vertices of Appendix A.3.1. Clipping places a weight on that boundary exactly: over the 53 runs the logged clipped fraction of a tensor has a median between 0.07% and 9.1% at the first epoch and between 0.001% and 4.5% at the last.

[3] For an inequality written as $-x_{i,e}^{(\ell)} \leq 0$, the KKT complementary-slackness condition is $\gamma_{i,e}^{(\ell)} x_{i,e}^{(\ell)*} = 0$. Thus, strict positivity of the primal variable forces the associated nonnegativity multiplier to vanish; see Bonnans and Shapiro (2000).

A.3.3 Regularizer Gradient with Respect to the Step Sizes

Holding $\bar{w}^{(\ell)}$ fixed, the step size a_ℓ appears explicitly in the full inner Lagrangian $\mathscr{L}_\ell^{\text{full}}$ only through the representation-constraint term

$$-\sum_{i\in\mathcal{I}_\ell}\mu_i^{(\ell)}\sum_{b\in\mathcal{B}}a_\ell q_b x_{i,b}^{(\ell)},$$

because $v_b^{(\ell)}(a_\ell)=a_\ell q_b$. The envelope theorem therefore computes the derivative of the optimal-value function by differentiating this explicit dependence at an optimal primal–dual solution; derivatives of the optimal assignments $x_{i,b}^{(\ell)*}$ with respect to a_ℓ are not required (Bonnans and Shapiro, 2000). Since $\partial v_b^{(\ell)}/\partial a_\ell = q_b$, this gives

$$\left.\frac{\partial\Phi_\ell}{\partial a_\ell}\right|_{\bar{w}^{(\ell)}} = -\sum_{i\in\mathcal{I}_\ell}\mu_i^{(\ell)}\sum_{b\in\mathcal{B}}q_b x_{i,b}^{(\ell)*} = -\frac{1}{a_\ell}\sum_{i\in\mathcal{I}_\ell}\mu_i^{(\ell)}\bar{w}_i^{(\ell)}. \tag{34}$$

Thus, the first equality follows from the envelope theorem. The second follows from the representation constraint in Eq. (4), together with $v_b^{(\ell)}=a_\ell q_b$. This is the derivative at fixed $\bar{w}^{(\ell)}$, and it is only one of the two channels through which the step size reaches Φ_ℓ: the second runs through the dependence of the clipped weight $\bar{w}_i^{(\ell)}(a_\ell)$ on a_ℓ, and it is derived, together with the assembled step-size direction, in Appendix A.3.4.

A.3.4 Total Training Gradients

Define the gradient of the task loss with respect to the quantized weight as

$$\delta_i^{(\ell)} := \frac{\partial\mathcal{L}_{\text{task}}}{\partial\widehat{w}_i^{(\ell)}}.$$

$\delta_i^{(\ell)}$ is the upstream gradient returned by backpropagation. The LSQ straight-through estimator (STE) approximates[4]

$$\frac{\partial\widehat{w}_i^{(\ell)}}{\partial w_i^{(\ell)}} \simeq \mathbf{1}\{Q_n \le r_i^{(\ell)} \le Q_p\}, \tag{35}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, equal to one when its argument is true and zero otherwise, and $r_i^{(\ell)}$ is defined in Eq. (14). Under this estimator, the latent-weight component of the task-loss gradient is

$$\nabla_{w_i^{(\ell)}}\mathcal{L}_{\text{task}} \simeq \delta_i^{(\ell)}\mathbf{1}\{Q_n \le r_i^{(\ell)} \le Q_p\}. \tag{36}$$

We next derive the LSQ gradient with respect to the step size. If $r_i^{(\ell)} < Q_n$ or $r_i^{(\ell)} > Q_p$, clipping gives respectively $\widehat{w}_i^{(\ell)} = a_\ell Q_n$ or $\widehat{w}_i^{(\ell)} = a_\ell Q_p$; the corresponding derivatives with respect to a_ℓ are therefore Q_n and Q_p. Inside the representable interval,

$$\widehat{w}_i^{(\ell)} = a_\ell\,\mathrm{round}(r_i^{(\ell)}).$$

Using the STE surrogate $\partial\,\mathrm{round}(r)/\partial r \simeq 1$ gives

$$\frac{\partial\widehat{w}_i^{(\ell)}}{\partial a_\ell} \simeq \mathrm{round}(r_i^{(\ell)}) + a_\ell\frac{\partial r_i^{(\ell)}}{\partial a_\ell} = \mathrm{round}(r_i^{(\ell)}) - r_i^{(\ell)}.$$

These three cases are collected in the function

$$h(r) := \begin{cases} Q_n, & r < Q_n, \\ \mathrm{round}(r) - r, & Q_n \le r \le Q_p, \\ Q_p, & r > Q_p, \end{cases} \tag{37}$$

[4]The quantizer contains a nondifferentiable rounding operation. During QAT, the STE leaves the quantized forward pass unchanged but replaces the derivative of the quantizer in the backward pass with a surrogate derivative, thereby enabling gradient-based optimization; see Bengio et al. (2013); Esser et al. (2020).

where $r \in \mathbb{R}$ is a scalar argument. Applying the chain rule and the definition of $\delta_i^{(\ell)}$ gives

$$\frac{\partial \mathcal{L}_{\text{task}}}{\partial a_\ell} \simeq \sum_{i \in \mathcal{I}_\ell} \frac{\partial \mathcal{L}_{\text{task}}}{\partial \widehat{w}_i^{(\ell)}} \frac{\partial \widehat{w}_i^{(\ell)}}{\partial a_\ell} = \sum_{i \in \mathcal{I}_\ell} \delta_i^{(\ell)} h(r_i^{(\ell)}).$$

Accordingly, the step-size component of the task-loss gradient is

$$\nabla_{a_\ell} \mathcal{L}_{\text{task}} \simeq \sum_{i \in \mathcal{I}_\ell} \delta_i^{(\ell)} h(r_i^{(\ell)}). \tag{38}$$

We must additionally account for the dependence of $\bar{w}_i^{(\ell)}(a_\ell)$ on the step size through the clipping operation in Eq. (15). Selected latent-weight and step-size components of the PRESTO subdifferential are

$$\left(\nabla_{\text{PRESTO}}\right)_{w_i^{(\ell)}} := \beta_i^{(\ell)*} \mathbf{1}\{Q_n \le r_i^{(\ell)} \le Q_p\}, \tag{39}$$

$$\left(\nabla_{\text{PRESTO}}\right)_{a_\ell} := -\frac{1}{a_\ell} \sum_{i \in \mathcal{I}_\ell} \mu_i^{(\ell)} \bar{w}_i^{(\ell)} + \sum_{\substack{i \in \mathcal{I}_\ell \\ r_i^{(\ell)} < Q_n}} \beta_i^{(\ell)*} Q_n + \sum_{\substack{i \in \mathcal{I}_\ell \\ r_i^{(\ell)} > Q_p}} \beta_i^{(\ell)*} Q_p. \tag{40}$$

The first term in Eq. (40) is the direct dependence of Φ_ℓ on a_ℓ at fixed $\bar{w}^{(\ell)}$, given by the step-size envelope derivative of Appendix A.3.3. There is also an indirect dependence through the clipped weights $\bar{w}_i^{(\ell)}(a_\ell)$. Holding the latent weight $w_i^{(\ell)}$ fixed, Eq. (15) gives

$$\frac{\partial \bar{w}_i^{(\ell)}}{\partial a_\ell} = \begin{cases} Q_n, & r_i^{(\ell)} < Q_n, \\ 0, & Q_n \le r_i^{(\ell)} \le Q_p, \\ Q_p, & r_i^{(\ell)} > Q_p. \end{cases}$$

Indeed, inside the representable interval $\bar{w}_i^{(\ell)} = w_i^{(\ell)}$ and hence does not vary with a_ℓ; outside it, the clipped weight equals respectively $a_\ell Q_n$ or $a_\ell Q_p$. Since $\beta_i^{(\ell)*}$ is the selected subgradient of Φ_ℓ with respect to $\bar{w}_i^{(\ell)}$ at fixed a_ℓ, the subgradient chain rule contributes

$$\sum_{i \in \mathcal{I}_\ell} \beta_i^{(\ell)*} \frac{\partial \bar{w}_i^{(\ell)}}{\partial a_\ell}.$$

Substituting the preceding piecewise derivative produces the final two sums in Eq. (40).

Equations (36) and (38) provide the latent-weight and step-size components of the task-loss gradient of Section 2.2. Collecting these components gives

$$\nabla_{(w,a,\theta_{\text{fp}})} \mathcal{L}_{\text{task}} = \left(\left(\nabla_{w_i^{(\ell)}} \mathcal{L}_{\text{task}}\right)_{\ell \in \Lambda,\ i \in \mathcal{I}_\ell}, \left(\nabla_{a_\ell} \mathcal{L}_{\text{task}}\right)_{\ell \in \Lambda}, \nabla_{\theta_{\text{fp}}} \mathcal{L}_{\text{task}} \right). \tag{41}$$

Likewise, Eqs. (39) and (40) provide the corresponding components of the selected PRESTO subgradient introduced in Section 2.2, so

$$\nabla_{\text{PRESTO}} = \left(\left(\left(\nabla_{\text{PRESTO}}\right)_{w_i^{(\ell)}} \right)_{\ell \in \Lambda,\ i \in \mathcal{I}_\ell}, \left(\left(\nabla_{\text{PRESTO}}\right)_{a_\ell} \right)_{\ell \in \Lambda}, 0 \right). \tag{42}$$

Finally, combining Eqs. (41) and (42), the direction passed to the optimizers during training is

$$\nabla_{\text{train}} \mathcal{J} := \nabla_{(w,a,\theta_{\text{fp}})} \mathcal{L}_{\text{task}} + \nabla_{\text{PRESTO}}. \tag{43}$$

Up to the straight-through approximation of its task-loss components, this direction lies in $\partial_{(w,a,\theta_{\text{fp}})} \mathcal{J}$ when the dual is at an optimum; with the truncated dual actually used it is the inexact proxy characterized in Appendix A.5.2.

A.3.5 Order of the Outer Update

Let k index the steps of the outer optimizer. At the beginning of step k, w_k denotes the collection of latent quantized weights, a_k the collection of learned step sizes, and $\theta_{\text{fp},k}$ the parameters retained in full precision. All quantities entering the gradients at step k must be evaluated from this same parameter state, in the following order:

(1) construct the levels $v_{b,k}^{(\ell)} = a_{\ell,k} q_b$;
(2) perform the forward pass using $\widehat{w}(w_k, a_k)$ and compute the task-loss gradients;
(3) solve the inner problem using $\bar{w}_k$, a_k, and the levels v_k, obtaining x_k^*, y_k^*, μ_k, and β_k^*;
(4) form the training direction of Appendix A.3.4 from the task-loss components in Eqs. (36) and (38) and the selected PRESTO components in Eqs. (39) and (40);
(5) update w_k, a_k, and $\theta_{\mathrm{fp},k}$ with their respective optimizers, producing the state indexed by $k+1$.

In this list, v_k collects all real-valued levels, $\bar{w}_k$ collects all clipped latent weights, and x_k^*, y_k^*, μ_k, and β_k^* collect the corresponding solutions or derivatives across all tensors and weights. In particular, the dual solver at step k must use the levels generated by a_k, because those same levels produced the quantized weights in the forward pass. Updating a_k before solving the inner problem would instead combine the clipped weights and task gradients associated with one codebook with PRESTO gradients associated with another. The resulting quantity would therefore not be a gradient of the single parameter state appearing in Eq. (8).

A.4 The Dual Solver

The multiplier is not solved for but advanced, a few warm-started steps per call. This section states the iteration and then the property of its step size that decides whether the entropy coefficient has any effect at all.

A.4.1 Inner Dual Iteration

Let N_{dual} be the positive integer specifying the number of dual iterations performed per solver call, let $t \in \{0, \dots, N_{\mathrm{dual}} - 1\}$ index those iterations, and let $\eta_\xi > 0$ be the dual step size. At the multiplier vector $\xi^{(\ell,t)}$, the implementation first solves the per-weight and count subproblems and forms the normalized supergradient

$$g_b^{(\ell,t)} := \frac{\widehat{c}_b^{(\ell,t)} - c_b^{(\ell,t)*}}{c_{\max}}, \qquad b \in \mathcal{B}. \tag{44}$$

If the per-weight subproblems assign more mass to symbol b than the count subproblem does, then $g_b^{(\ell,t)} > 0$ and the corresponding multiplier is pushed upwards by dual ascent. Division by $c_{\max}$ normalizes the residual by the largest count the box admits; with the value $c_{\max} = N$ of our runs this is one constant for the whole network, so the residual of a tensor, and with it its update, still scales with the number of its weights (Appendix A.2.2).

The dual solver used in our experiments applies a FISTA-style momentum extrapolation (Beck and Teboulle, 2009); it is not an optional alternative to the update above. To state the implemented iteration precisely, let $\tau_0 = 1$ and initialize $\xi^{(\ell,-1)} = \xi^{(\ell,0)}$; the multipliers are warm-started from one call to the next, but this momentum state is reset at every call. At iteration t, the momentum parameter and extrapolated multiplier vector are

$$\tau_{t+1} := \frac{1 + \sqrt{1 + 4\tau_t^2}}{2}, \tag{45}$$

$$\widetilde{\xi}^{(\ell,t)} := \xi^{(\ell,t)} + \frac{\tau_t - 1}{\tau_{t+1}} \left(\xi^{(\ell,t)} - \xi^{(\ell,t-1)} \right). \tag{46}$$

The normalized supergradient in Eq. (44) is then applied to the extrapolated point[5]:

$$z_b^{(\ell,t+1)} := \widetilde{\xi}_b^{(\ell,t)} + \eta_\xi g_b^{(\ell,t)}, \qquad b \in \mathcal{B}. \tag{47}$$

The count bounds in Eq. (28), together with the unconstrained count solution of Appendix A.2.2, correspond to the multiplier bounds

$$\xi_{\min}^{(\ell)} := \frac{T_3}{\ln(2)} \big(\ln c_{\min} + 1 \big), \qquad \xi_{\max}^{(\ell)} := \frac{T_3}{\ln(2)} \big(\ln c_{\max} + 1 \big). \tag{48}$$

[5]The plus sign follows from the fact that Eq. (12) is a maximization problem, so the update performs supergradient ascent.

After Eq. (47), every component is clipped to this interval:

$$\xi^{(\ell,t+1)} := \mathrm{clip}\Big(z^{(\ell,t+1)}; \xi^{(\ell)}_{\min}, \xi^{(\ell)}_{\max}\Big)\,. \tag{49}$$

Here, clip acts componentwise, so each $\xi^{(\ell)}_b$ remains inside $[\xi^{(\ell)}_{\min}, \xi^{(\ell)}_{\max}]$. The per-weight subproblems, the count solution, the residual, the extrapolation and the final clipping are repeated for the prescribed number of dual iterations N_{dual}.

This is a FISTA-style rather than a literal canonical FISTA iteration: the supergradient $g^{(\ell,t)}$ is evaluated at $\xi^{(\ell,t)}$, whereas its correction is applied to the extrapolated vector $\widetilde{\xi}^{(\ell,t)}$. Stating both evaluation points is necessary to reproduce the implemented update unambiguously.

Once the N_{dual} dual iterations are completed, the per-weight solutions x^* and y^*, together with the associated quantities μ and β^*, computed during the last iteration are used in the PRESTO-gradient formulas of Section 3.2.[6]

A.4.2 Dual Step Size and Convergence Rate

The entropy coefficient reaches the latent weights through one channel only. The per-weight subproblem is solved against the raw multipliers $\xi^{(\ell)}_b$, and T_3 does not appear in it: it enters the dual decomposition only through the count subproblem. The entropy term is therefore exactly as strong as the dual is converged, and no stronger.

This creates a scale coupling that is easy to miss. The residual of Eq. (44) is normalized to $[-1, 1]$, so an absolute step η_ξ moves the multiplier by a fixed distance per iteration. The interval in which the multiplier is useful, however, has width $T_3 \log_2(c_{\max}/c_{\min})$, where $c_{\max}$ is the upper clamp on the relaxed counts, set in the runs of this paper to $N := |\theta|$, the number of parameters of the whole network, quantized and floating-point alike; the width is therefore proportional to T_3. Raising the coefficient therefore moves the target away at the same rate at which it raises it, and we measured that effect directly.

Let one *traversal* denote the number of times the multiplier could cross its own interval in one epoch, that is $\eta_\xi\, \bar{g}\, N_{\mathrm{dual}} S/(T_3 \log_2(N/c_{\min}))$, where S is the number of dual calls per epoch and $\bar{g} \approx 1/C$ is a nominal residual, that of a tensor holding all N parameters; a tensor of n_ℓ weights traverses n_ℓ/N times as fast, so Table 2 compares runs on a common nominal scale. Table 2 reports it for four runs made with an absolute step of 3×10^{-9}, before we adopted the relative one, and for the EfficientNet-B0 run that adopted it.

Table 2: Traversals of the dual interval per epoch.

Architecture	T_3	Dual calls / epoch	Traversals / epoch	Outcome
ResNet-18	3×10^{-8}	313	0.195	entropy term effective
DeiT-Small	1.5×10^{-8}	156	0.189	entropy term effective
EfficientNet-B0	3×10^{-7}	78	0.0050	inert
EfficientNet-B0	3×10^{-6}	78	0.0005	inert
EfficientNet-B0	3×10^{-8}	78	0.190	effective

The two inert rows differ tenfold in T_3, and the norms of the applied regularizer gradient at the first epoch differ by 0.013%, 2.756366×10^{-4} against 2.756720×10^{-4}. Neither compressed more than the LSQ baseline of the same network at the same epoch before they were stopped, after five and four epochs. The last row is on the network of those two, with the step expressed as a fraction of the interval, 1.3×10^{-2} per iteration, a value chosen so that the traversal rate of the two effective rows is recovered at the seventy-eight dual calls per epoch that this network affords: it lands at 0.190 against their 0.195 and 0.189, and the entropy term works again, with about four times the LSQ baseline's

[6] A finite N_{dual} does not, in general, produce the exact maximizer of the dual problem in Eq. (12). It is a computational budget that trades the quality of the approximate dual solution against the cost of each training update, and is therefore an important hyperparameter. To mitigate the use of a small budget, each dual-solver call is warm-started from the multiplier vector returned by the preceding call rather than from a new initialization. This allows progressive refinement across calls, although the underlying weights and codebook may change between them. The dependence on N_{dual} and the effect of warm starting are discussed in Appendices A.5.2 and B.5.

change in packet size, symbol entropy and sparsity, at 0.14 accuracy points. It is not a one-variable comparison against the two inert rows, which also differ in T_3, in budget and in the shape of the dose schedule. What the table supports is the association between the traversal rate and the outcome: the three rows near 0.19 are the three in which the entropy term acts, and the two that are about 40 and 400 times below it are the two in which it does not.

The practical rule is that η_ξ should be expressed as a fraction of the dual interval rather than as an absolute distance. It then follows T_3 without further tuning; the number of dual calls per epoch still has to be accounted for, as the EfficientNet-B0 row does, and under $c_{\max} = N$ so does the size of the tensor. At the 1.3×10^{-2} we use in every relative-step run, the rate per epoch runs from 0.19 on EfficientNet-B0 to 1.53 on the ViT-B/16 rows at $N_{\mathrm{dual}} = 6$, all at or above the value at which the term acts.

A.5 Guarantees

The relaxation is exact for the problem as stated. The implementation, however, poses a slightly smaller problem and stops the dual before its maximum, and the two subsections below state each of those facts precisely.

A.5.1 Strong Duality of the Relaxation

Proposition 1 (No duality gap)**.** *Fix a tensor ℓ, a step size $a_\ell > 0$ and a clipped weight vector $\bar{w}^{(\ell)}$ satisfying $|\bar{w}_i^{(\ell)}| \le V_i^{\max}$ for every $i \in \mathcal{I}_\ell$. Then the inner problem of Eq.* (3) *has a finite optimal value, its minimum is attained, and the partial dual obtained by relaxing only the count constraints* (6) *has no gap:*

$$\Phi_\ell(\bar{w}^{(\ell)}, a_\ell) = \sup_{\xi^{(\ell)} \in \mathbb{R}^B} \varphi^{(\ell)}_{\bar{w}^{(\ell)}, a_\ell}(\xi^{(\ell)}).$$

If, in addition, at least one weight lies strictly inside its representable interval, the supremum is attained.

Proof. Placing all of the mass of each non-zero weight on the extreme level attaining $V_i^{\max}$, and none for a zero weight, gives a feasible point. The feasible set is compact, since $0 \le x_{i,b}^{(\ell)} \le y_i^{(\ell)} \le 1$ and the counts are sums of the masses, and the objective is closed and bounded below, since $c \log_2 c \ge -1/(e \ln 2)$; the minimum is therefore attained. Let $p(u)$ be the optimal value when the count constraints are perturbed to $c_b^{(\ell)} = \sum_{i \in \mathcal{I}_\ell} x_{i,b}^{(\ell)} - u_b$, a convex function because the perturbed problem is jointly convex in the variables and in u. For u near the origin the feasible sets lie in a common compact set and form a closed graph, so minimizers for a sequence $u_k \to 0$ have a limit point that is feasible at $u = 0$, and lower semicontinuity of the objective gives $p(0) \le \liminf_k p(u_k)$. Hence p is lower semicontinuous at the origin and never $-\infty$, so its closure agrees with it there, and the absence of a gap follows from Rockafellar (1974, Thm. 15). If some weight lies strictly inside its representable interval, it can place positive mass on every symbol, so every small perturbation u remains feasible with a bounded objective; p is then bounded above on a neighborhood of the origin, and the supremum is attained by Rockafellar (1974, Thm. 17). □

Because the minimum defining $\varphi^{(\ell)}$ is attained over a compact set, Danskin's theorem (Danskin, 1966) applies, and the derivatives of Appendices A.3.1 and A.3.3 are envelope derivatives at the inner optimum rather than derivatives of a smoothed surrogate. They are gradients wherever the inner minimizer is unique, which holds in y whenever $T_1 > 0$ and $\bar{w}_i^{(\ell)} \neq 0$. At a lower-hull breakpoint or at the representability boundary the minimizer need not be unique, and they are then the subgradient selected there, with the exception at interior vertices stated in Appendix A.3.1.

The implementation solves a boxed version of this problem and advances the multiplier by a few warm-started steps rather than maximizing, so it reaches neither the optimal multipliers nor, in general, the value above; Appendix A.5.2 gives the boxed version of Proposition 1 and characterizes what the training direction is exact for.

A.5.2 Boxed Problem and Truncated Dual

The inner problem of Eq. (3) lets $c_b^{(\ell)}$ range over $[0, \infty)$, whereas the implementation confines it to $[c_{\min}, c_{\max}]$ with $c_{\min} = 10^{-2}$ and projects the multiplier onto the matching interval of Eq. (48). That box is not cosmetic and we do not treat it as such. Write $\Phi_\ell^\square$ for the inner problem of Eq. (3) with $c^{(\ell)} \in [c_{\min}, c_{\max}]^B$ imposed, and $\varphi_\ell^\square(\xi^{(\ell)})$ for the partial dual obtained from it by relaxing the count constraints exactly as before. Restricting the feasible set can only raise the minimum, so $\Phi_\ell^\square \geq \Phi_\ell$, and what the solver works with is $\varphi_\ell^\square$, not $\varphi^{(\ell)}$; the two problems coincide when the box is inactive. The multiplier is boxed as well. Writing

$$\Xi_\ell := \left[\xi_{\min}^{(\ell)}, \xi_{\max}^{(\ell)}\right]^B$$

for the interval of Eq. (48), onto which Eq. (49) projects after every step, the iteration is an ascent on $\varphi_\ell^\square$ *restricted to* Ξ_ℓ, so the value it can reach is $\max_{\xi \in \Xi_\ell} \varphi_\ell^\square(\xi)$, which by weak duality satisfies

$$\max_{\xi \in \Xi_\ell} \varphi_\ell^\square(\xi) \;\leq\; \max_{\xi \in \mathbb{R}^B} \varphi_\ell^\square(\xi) \;=\; \Phi_\ell^\square,$$

with equality when the projection is inactive at a dual optimum. Since $\xi_{\min}^{(\ell)}$ and $\xi_{\max}^{(\ell)}$ are exactly the multipliers at which the unconstrained count solution meets $c_{\min}$ and $c_{\max}$, the projection is the dual face of the count box rather than a second, independent restriction; but it can bind, and we report below how often it does.

Proposition 1 carries over to $\Phi_\ell^\square = \max_{\xi^{(\ell)} \in \mathbb{R}^B} \varphi_\ell^\square(\xi^{(\ell)})$, the unprojected statement, but its hypothesis has to be strengthened, because the box shrinks the feasible set and feasibility must be re-established. A point that places all of a weight's mass on one extreme level, feasible for Φ_ℓ, leaves every other symbol at count zero, which the box forbids. Nor does it suffice that a weight lie strictly inside the representable interval, since a weight arbitrarily close to an endpoint does admit a strictly positive representation, but with coefficients that vanish as it approaches that endpoint, so it may be unable to carry $c_{\min}$ on every symbol while keeping $y_i^{(\ell)} \leq 1$. What suffices is a margin, and for a uniform codebook the margin can be written down. Give every symbol a floor ε, which consumes mass $B\varepsilon$ and represents the value $\varepsilon a_\ell \sum_{b \in \mathcal{B}} q_b$. The mass left over, $1 - B\varepsilon$, can then be placed on any single level, so a weight is representable with every coefficient at least ε whenever

$$\varepsilon \sum_{b \in \mathcal{B}} q_b + (1 - B\varepsilon)\, Q_n \;\leq\; \frac{\bar{w}_i^{(\ell)}}{a_\ell} \;\leq\; \varepsilon \sum_{b \in \mathcal{B}} q_b + (1 - B\varepsilon)\, Q_p.$$

The endpoints are attained by putting all of the leftover mass on one extreme level, and every interior value by a convex combination or by using less of it. With our grid, $Q_n = -8$, $Q_p = 7$, $B = 15$ and $\sum_b q_b = -8$, taking $\varepsilon = 2c_{\min} = 2 \times 10^{-2}$ gives $\bar{w}_i^{(\ell)}/a_\ell \in [-5.76,\, 4.74]$, and one weight in that band places strictly more than $c_{\min}$ on each of the B symbols, which is what the boxed problem needs in order to have a feasible point at all. The existence of such a weight is the hypothesis under which Proposition 1 carries over to the boxed problem. We state it as a hypothesis and not as an established property of our runs: it is checkable in one pass over a tensor, it concerns a single weight out of millions, and it fails only for a tensor whose entire weight vector sits outside $[-5.76\, a_\ell,\, 4.74\, a_\ell]$ on its own grid fitted by mean squared error (MSE). We did not instrument the runs to certify it, and nothing we compute depends on it, since the exactness statement below is made at the multiplier the solver holds.

The box binds, so the distinction is not a formality. Averaged over the dual calls of an epoch, the fraction of multipliers within 10^{-6} of the width of the dual interval from a bound is largest in the first epoch in which the dual runs, where it reaches 0.003 on ResNet-18 at a hundred epochs and 0.064 in the three repeats of its Table 1 configuration, 0.09 on ViT-B/16, 0.31 on ResNet-50 and 0.56 on AlexNet, and it falls to zero by the last epoch of every PRESTO run of Tables 1 and 8. What we ascend is therefore the projected dual of the boxed problem, and every claim of exactness below is made against it.

Exactness of the training direction. The boxed version of Proposition 1 guarantees that

$$\Phi_\ell^\square(\bar{w}^{(\ell)}, a_\ell) = \max_{\xi^{(\ell)} \in \mathbb{R}^B} \varphi_\ell^\square(\xi^{(\ell)}) \;\geq\; \max_{\xi^{(\ell)} \in \Xi_\ell} \varphi_\ell^\square(\xi^{(\ell)}),$$

and the implementation attains neither maximum: it advances the multiplier inside Ξ_ℓ by N_{dual} warm-started supergradient steps per call, with $N_{\mathrm{dual}} = 3$ in every run of this paper except the two ViT-B/16 frontier points, which use 6. Writing ξ for the multiplier the solver currently holds and $\xi^\star$ for a dual optimum, two statements have to be kept apart. The first is exact up to the selection at interior vertices of Appendix A.3.1: at the fixed ξ the per-weight and count subproblems are solved to optimality, so the quantities of Appendices A.3.1 and A.3.3 belong to $\partial_{(\bar{w}^{(\ell)}, a_\ell)} \varphi_\ell^{\square}(\xi)$, the subdifferential of the boxed relaxed value at that multiplier, and $\varphi_\ell^{\square}(\xi)$ is the number the solver computes, not the maximized dual value. The second is not exact. By weak duality $\varphi_\ell^{\square}(\xi) \leq \max_{\Xi_\ell} \varphi_\ell^{\square} \leq \Phi_\ell^{\square}$ for every $\xi \in \Xi_\ell$, with equality throughout only at an unprojected dual optimum, so those same quantities are in general *not* elements of $\partial \Phi_\ell^{\square}$. The outer optimization is therefore a descent on a Lagrangian lower bound of the boxed problem, driven by a direction that is an exact subgradient of that lower bound and only an inexact proxy for $\partial \Phi_\ell^{\square}$, to which it belongs when the dual is optimal. Neither statement is made against Φ_ℓ itself, from which $\Phi_\ell^{\square}$ differs by the box. The gap between $\varphi_\ell^{\square}(\xi)$ and $\Phi_\ell^{\square}$ is diagnosed by the dual residual, the difference between the counts reconstructed from the per-weight solutions and the counts the symbol subproblems return, which is the supergradient the dual update consumes. A zero residual certifies dual optimality, but the converse does not follow, for two reasons. The dual is concave and may be nondifferentiable, so optimality asks only that zero belong to the superdifferential, and the residual produced by one selected primal minimizer need not vanish. A multiplier held at a bound by the projection of Eq. (48) can also be stationary for the projected dual with a non-zero residual. The residual is therefore a diagnostic and not a certificate, and we do not claim an error bound.

Two properties of the scheme keep that inexactness from being a problem in practice, and one measurement shows what happens when they fail. The multiplier is warm-started across calls and the weights move slowly between calls, so the dual is not restarted at every step but tracks a slowly moving optimum, which is the standard setting of an incremental Lagrangian method rather than a fresh truncated solve. The dual step is expressed as a fraction of the dual range, so the number of times the multiplier can cross that range per epoch does not depend on T_3; it still scales with the number of dual calls per epoch and, under $c_{\max} = N$, with the size of the tensor, and Table 2 reports it for the runs that exposed the problem. When that second property was absent the failure was visible and severe: with an absolute step, raising T_3 tenfold left the applied gradient within 0.013% of itself and the model compressed no more than its LSQ baseline, which is the negative result reported in Appendix C.9. We therefore report the traversal rate rather than claim dual convergence.

A.6 Effect of the Entropy Term

Write $m = \sum_{b \in \mathcal{B}} c_b^{(\ell)}$ for the non-zero mass and $\pi_b = c_b^{(\ell)}/m$ for the conditional distribution over $\mathcal{B}$. The ideal zero-order code length of the tensor decomposes as

$$n_\ell H(m/n_\ell) + m\, H(\pi), \tag{50}$$

the support mask plus the value stream, which is n_ℓ times the entropy of the distribution over all C_ℓ symbols, the zero symbol included. This is a proxy and not the size the coder produces: a real coder carries container overhead and can exploit structure that a symbol histogram does not describe, which is why the mask costs 2.905% where the proxy predicts 3.12% (Appendix C.7). The value stream goes the same way on AlexNet, where the zero-order entropy of the non-zero symbols is 1.686 bits while the coder delivers those symbols at 1.34 bits, the difference being structure that a single network-wide histogram does not describe, per-tensor statistics included. Every size we report for our own runs is a measurement; the proxy is what the objective can be written against. The entropy term acts on both channels through the non-zero mass m. The scalar map $c \mapsto c \log_2 c$ is minimized at $c = 1/e$, so the unconstrained minimizer of $\sum_b c_b^{(\ell)} \log_2 c_b^{(\ell)}$ assigns a count of $1/e$ to every non-zero symbol, an $O(1)$ occupancy that is negligible against a tensor of n_ℓ weights. The penalty therefore prices membership of $\mathcal{B}$ and drives mass into the excluded zero symbol, which shortens the value stream $m\, H(\pi)$ and, past half sparsity, the mask $n_\ell H(m/n_\ell)$ as well; at a fixed conditional distribution the total falls whenever $H(\pi) > \log_2\big(m/(n_\ell - m)\big)$, which holds, on the statistics pooled over each network, at the end of every run of Tables 1, 6 and 8. At a fixed m it acts in the opposite direction, because the same sum is convex and symmetric and is therefore minimized at uniform counts, and the identity $mH(\pi) = m \log_2 m - \sum_b c_b^{(\ell)} \log_2 c_b^{(\ell)}$ makes that explicit: at fixed mass, minimizing our term maximizes $mH(\pi)$. The two effects are of different size, $m \log_2 m$

against $m \log_2 B_\ell$ with $B_\ell = 15$ and m in the millions, so the mass channel is the larger by a factor of five to six, which is the direction the term ablation measures. We state this because the reduction we report in the entropy of the non-zero symbols is an empirical consequence of the sparsity the term creates and not a property of the objective; a term that directly minimized $mH(\pi)$ would be concave in the counts, and neither the decomposition of Section 3.1 nor Proposition 1 would survive it.

A.7 Algorithms and Additional Diagnostics

Algorithm 1 Exact inner solve for one tensor at a fixed multiplier. Cost $O(C^2 + n_\ell C)$; no iteration.

Require: clipped weights $\bar{w} \in \mathbb{R}^{n_\ell}$, levels $v_b = a_\ell q_b$ for $b \in \mathcal{B}$, multiplier $\xi \in \mathbb{R}^B$, coefficients T_1, T_2

Ensure: optimal masses y^*, relaxed counts $\widehat{c}$, weight subgradients β^*

build the lower convex hull of $\{(v_b, \xi_b)\}_{b \in \mathcal{B}}$ by a left-to-right scan of the sorted levels ▷ $O(C^2)$ at most, once per tensor per dual iteration
for each weight i **do**
  $V_i^{\max} \leftarrow \max\{|v_b| : \bar{w}_i v_b \geq 0\}$; $y_i^{\min} \leftarrow |\bar{w}_i| / V_i^{\max}$ (and 0 if $\bar{w}_i = 0$)
  $\mathcal{Y} \leftarrow \{y_i^{\min}, 1\} \cup \{\, |\bar{w}_i| / |v_{(k)}| \,\} \cap [y_i^{\min}, 1]$ ▷ facet breakpoints, Eq. (25)
  for each subinterval $[y_-, y_+]$ induced by $\mathcal{Y}$, in order **do**
    read (μ_i, s_i) of the facet active there ▷ Eq. (21)
    if $s_i + T_2 > 0$ **then** $y^{\text{cand}} \leftarrow |\bar{w}_i| \sqrt{T_1 / (s_i + T_2)}$ ▷ Eq. (24)
    end if
    evaluate G_i at y^{cand} if it lies in $[y_-, y_+]$, and at y_- and y_+ ▷ Eq. (23)
  end for
  $y_i^* \leftarrow \arg\min$ over all evaluated points; $z_i^* \leftarrow 1 - y_i^*$
  place the mass on the at most two hull-adjacent symbols that represent $\bar{w}_i$ at y_i^*; accumulate into $\widehat{c}$
  if $\bar{w}_i = 0$ **then** $y_i^* \leftarrow 0$; $\beta_i^* \leftarrow 0$ ▷ the ratio is $0/0$; see Appendix A.2.1
  else $\beta_i^* \leftarrow \mu_i + 2T_1 \bar{w}_i / y_i^*$, or the boundary selection of Appendix A.3.2 if $y_i^* = y_i^{\min}$, i.e. if $\bar{w}_i / y_i^*$ is the extreme level ▷ Eq. (13)
  end if
end for

Algorithm 1 states the exact per-weight solver of Appendix A.2.1 and Algorithm 2 the training loop that calls it. The two are written for a single tensor and a single optimizer step respectively; in the implementation the per-weight loop of Algorithm 1 is a vectorized sweep over the weights of the tensor, and Algorithm 2 runs under distributed data parallelism (DDP) with the dual state replicated.

Algorithm 2 One PRESTO optimizer step: the regularizer enters the same backward pass as the task loss.

Require: latent weights w, step sizes a, warm-started multipliers ξ, step counter k

$\widehat{w} \leftarrow \mathrm{LSQ}(w, a)$; accumulate $\nabla \mathcal{L}_{\mathrm{task}}$ by backpropagation with the straight-through estimator
if $T_3 > 0$ **and** $k \bmod \texttt{entropy_every} = 0$ **and** the entropy warm-up is over **then**
  for $t = 0$ **to** $N_{\mathrm{dual}} - 1$ **do** ▷ FISTA-style ascent, Appendix A.4.1
    $(y^*, \widehat{c}, \beta^*) \leftarrow$ Algorithm 1 at $\xi^{(t)}$ ▷ solved at the current multiplier
    $c_b^* \leftarrow \mathrm{clip}\big(2^{\xi_b^{(t)}/T_3 - 1/\ln 2}, c_{\min}, c_{\max}\big)$ ▷ count subproblem, closed form
    $g \leftarrow (\widehat{c} - c^*)/c_{\max}$
    $\widetilde{\xi} \leftarrow \xi^{(t)} + \frac{\tau_t - 1}{\tau_{t+1}}(\xi^{(t)} - \xi^{(t-1)})$ ▷ extrapolation *after* the solve
    $\xi^{(t+1)} \leftarrow \mathrm{clip}\big(\widetilde{\xi} + \eta_\xi g,\ \xi_{\min}, \xi_{\max}\big)$ ▷ η_ξ a fraction of $\xi_{\max} - \xi_{\min}$
  end for
  add β^* to the weight gradient and the matching envelope term to the step-size gradient ▷ Appendix A.3.4
else if $T_3 = 0$ **or** the entropy warm-up is not over **then**
  add the closed-form gradient of the T_1 and T_2 terms, which reduces to $2T_1\bar{w}$ on the in-range weights when $T_2 = 0$, the step sizes then receiving nothing from the regularizer
else
  add nothing: after the warm-up, with $T_3 > 0$ the regularizer reaches the weights only on the steps that call the dual ▷ see Appendix C.2
end if
step the weight optimizer and the separate step-size optimizer; carry ξ to the next call

Bit breakdown. Every size in this paper decomposes into three streams, reported per run by the serializer and summarized here for one of the three ResNet-18 headline repeats: at 61.83% sparsity the support mask costs 2.755% of the FP32 tensor bits, the value stream 2.066%, and the metadata 0.000197%, which is one FP32 step size for each of the twenty-one tensors plus two grid endpoints that these runs charge and that the released serializer, which never reads them back, does not. The metadata term is negligible under per-tensor grids and becomes a hard floor under per-channel ones, 0.599% on EfficientNet-B0 with its eighty-two tensors, which is the quantity Appendix C.7 prices.

B Reproducibility and Protocol

This appendix describes the protocol of the experiments and what is needed to check that the comparisons are fair: how the method is trained, how every run is configured and serialized, what is held identical across architectures, how the three coefficients are calibrated, and what the whole thing costs.

B.1 Training Implementation

Schedules and teacher losses are choices of implementation rather than extra terms of the regularizer, so we keep them separate from the definition of PRESTO. We quantize the convolutional and linear weight tensors with a signed uniform LSQ codebook and leave the biases, the normalization parameters and, on the transformers, the class token and the positional embeddings in FP32. The step size of each tensor is initialized to the value, among a grid of candidates, that minimizes the mean squared error between the full-precision weights and their quantized version, and is then trained by a separate Adam optimizer without weight decay. The remaining parameters are optimized by stochastic gradient descent (SGD) with momentum 0.9 and weight decay 10^{-4} on the ResNets, AlexNet and EfficientNet-B0, and by Adam without weight decay on the two transformers. Every forward pass uses the fake-quantized weights, the latent weights are restored before the optimizers take their step, and the inner solver is warm-started from the previous call and runs once every four outer steps. The learning rate of the weights rises linearly during a warm-up of one epoch on AlexNet and two on EfficientNet-B0, holds its base value through the epoch that follows the later of that warm-up and the entropy warm-up, and then decays along a cosine towards an explicit floor, which the last epoch approaches without reaching; under the dose ramp of Appendix B.4 the cosine spans the ramp and the rate then stays at the floor. The entropy coefficient is zero during its warm-up and then enters

gradually: t epochs after the warm-up it is $T_3(1 - e^{-t}) + \frac{1}{8}T_3e^{-t}$, an eighth of T_3 in the first epoch and 88% of it two epochs later, whereas under the dose ramp T_2 and T_3 are both scaled by the ramp fraction. In every run the Adam rate of the step sizes follows the cosine multiplier of the weights, so the grid stops moving once the weights have. Four architectures are distilled from a frozen FP32 copy of their own checkpoint, the two transformers at $\alpha = 0.9$ and AlexNet and EfficientNet-B0 at $\alpha = 0.5$, while neither ResNet is, because their LSQ baselines already exceed the FP32 accuracy and a strong teacher anchor pulls them back down. The distilled loss is $(1 - \alpha)$ times the cross-entropy plus α times the Kullback–Leibler divergence from the teacher's to the student's class distribution, at temperature 1. Training images receive a random resized crop to 224 pixels and a horizontal flip; evaluation resizes them to 256 pixels (248 on DeiT-Small) and takes the central 224-pixel crop. Forward and backward passes, evaluation included, run under bfloat16 autocast, so every accuracy we report, that of the FP32 checkpoint included, is measured under it. Batch-normalization statistics are recalibrated over 50 local batches per rank before every evaluation on the two ResNets and EfficientNet-B0, the three networks here that carry them. Table 3 gives every setting per run and Table 4 the quantized fraction of each network, while Appendix B.3 separates the parts of this training recipe that are held identical across architectures from the parts that are not.

B.2 Configurations, Checkpoints and Serialization

Table 3: Configuration of the runs of Tables 1, 6, 8 and 10. "rel." is the step-size rate expressed as a fraction of each tensor's own initial step; "abs." is a single absolute value; "headline" is the configuration in the first bold row of the network's block in Table 1.

Architecture	Role	Epochs	Batch	Weight LR, floor	Step-size LR	Distill.	(T_1, T_2, T_3)
ResNet-18	LSQ baseline	60	1024	10^{-2}, 1%	10^{-5} abs.	no	$(0, 0, 0)$
	ablation, 3 arms	60	1024	10^{-2}, 1%	10^{-5} abs.	no	see Appendix C.2
	headline, 3 repeats	60	1024	10^{-2}, 1%	10^{-5} abs.	no	$(10^{-5}, 10^{-7}, 6{\cdot}10^{-8})$
	frontier, 2 points	100	1024	10^{-2}, 1%	10^{-5} abs.	no	$T_3 \in \{10^{-7}, 2.6{\cdot}10^{-7}\}$
	sixty-epoch point	60	1024	10^{-2}, 1%	10^{-5} abs.	no	$(10^{-5}, 10^{-7}, 10^{-7})$
	joint and staged, 6 runs	20, 40	1024	10^{-2}, 1%	10^{-5} abs.	no	$(10^{-5}, 10^{-7}, 3{\cdot}10^{-8})$, Table 10
ResNet-50	LSQ baseline	20	1024	10^{-2}, 1%	10^{-5} abs.	no	$(0, 0, 0)$
	frontier, 2 points	20	1024	10^{-2}, 1%	10^{-5} abs.	no	$T_3 \in \{3{\cdot}10^{-8}, 10^{-7}\}$
DeiT-Small	LSQ baseline	25	2048	10^{-4}, 5%	10^{-5} abs.	0.9	$(0, 0, 0)$
	frontier, 4 points	25	2048	10^{-4}, 5%	10^{-5} abs.	0.9	$\kappa \in \{\frac{1}{5}, \frac{1}{2}, 1, 2\}$, both scaled
ViT-B/16	LSQ baseline	20	1024	$5{\cdot}10^{-5}$, 5%	10^{-5} abs.	0.9	$(0, 0, 0)$
	frontier, 2 points	20	1024	$5{\cdot}10^{-5}$, 5%	10^{-5} abs.	0.9	$\kappa \in \{\frac{1}{5}, 1\}$, both scaled
AlexNet	LSQ baseline	60	1024	$2{\cdot}10^{-3}$, 1%	10^{-3} rel.	0.5	$(10^{-5}, 0, 0)$
	ablation, 3 arms	60	1024	$2{\cdot}10^{-3}$, 1%	10^{-3} rel.	0.5	see Appendix C.2
EfficientNet-B0	LSQ baseline	30	4096	$2{\cdot}10^{-2}$, 5%	10^{-3} rel.	0.5	$(0, 0, 0)$
	PRESTO	30	4096	$2{\cdot}10^{-2}$, 5%	10^{-3} rel.	0.5	$(10^{-5}, 2.1{\cdot}10^{-7}, 6{\cdot}10^{-8})$

The reference dose of the transformers is $(T_2, T_3) = (5 \cdot 10^{-8}, 1.5 \cdot 10^{-8})$, and a common factor κ scales both. Two epoch counts warrant a word: ResNet-50 and ViT-B/16 run for twenty epochs against ResNet-18's sixty because their per-epoch cost under PRESTO is two to five times higher (Table 5), and the PRESTO runs of both networks in Table 1 ended with both axes still improving, so on them the shorter budgets understate rather than flatter the result. In the runs of Table 1, EfficientNet-B0 additionally gives eight bits to twenty-six tensors, its input convolution, its sixteen depthwise convolutions and nine layers of its first three stages, 3.75% of its weights, counted at full cost in every size; AlexNet gives eight bits to its first two convolutions, 0.54% of its weights.

Table 4 records which part of each checkpoint the packet covers, since every size in the paper is converted through model $= rf + (1 - f)$. The quantized set is every convolutional and linear weight tensor; what stays in floating point is the biases, the normalization vectors, the class token and the positional embeddings. On ViT-B/16 the set needs one remark, because torchvision stores the packed query, key and value projection of each block in `in_proj_weight`, a parameter that is not a module weight: quantizing only the tensors that look like module weights would cover 75.15% of that network instead of 99.68% and would make every size incomparable with the published ones.

Checkpoints. Every run starts from a public checkpoint of ResNet-18 or ResNet-50 (He et al., 2016), AlexNet (Krizhevsky et al., 2012), DeiT-Small (Touvron et al., 2021), ViT-B/16 (Dosovitskiy

Table 4: Quantized fraction f of each checkpoint, with the number of tensors and parameters it covers.

Architecture	Tensors	Quantized parameters	f [%]
ResNet-18	21	11,678,912	99.909
ResNet-50	54	25,502,912	99.788
DeiT-Small	50	21,912,576	99.374
ViT-B/16	50	86,292,480	99.682
AlexNet	8	61,090,496	99.983
EfficientNet-B0	82	5,236,192	99.010

et al., 2021) or EfficientNet-B0 (Tan and Le, 2019), identified by the content hash carried in its file name:

Architecture	Checkpoint
ResNet-18	`resnet18-f37072fd`
ResNet-50	`resnet50-0676ba61`
DeiT-Small	`deit_small_patch16_224-cd65a155`
ViT-B/16	`vit_b_16-c867db91`
AlexNet	`alexnet-owt-7be5be79`
EfficientNet-B0	`efficientnet_b0_rwightman-7f5810bc`

The serialized package is produced by one fixed procedure for every run and every architecture. The support is a packed bit mask, one bit per weight; the values are the integer symbol indices of the non-zero weights, four bits each, or eight on the eight-bit tensors, with at most four bits of padding per tensor; and the metadata is one FP32 step size per tensor, or one per output channel where per-channel grids are used. The mask and the value stream are compressed with `zstd` at level 22 with multithreading enabled, which is the only lossless coder used anywhere in this paper and is applied identically to every run; the metadata is not compressed and is charged at its raw cost of 32 bits per stored step size. The dual is updated every four optimizer steps with $N_{\mathrm{dual}} = 3$ iterations, except on the two ViT-B/16 frontier points where it is 6, a value the ViT-B/16 baselines also carry although their entropy term is off, and the LSQ step sizes use their own optimizer and learning rate, reported per run in Table 3. Two quantities of the dual are fixed for every run and are not tuned. The first is the lower clamp on the relaxed counts, $c_{\min} = 10^{-2}$, which fixes the interval onto which the multiplier is projected, derived in Appendix A.4.1. The second is the initialization: the multipliers of a tensor are drawn uniformly from $[0, 1]$, above that interval, so the first dual call projects them onto its upper bound and the momentum of its second iteration carries them to the lower one; from there they are warm-started from one call to the next. Training runs under Python 3.11.6 and PyTorch 2.2.0 (build `2.2.0a0+git6c8c5ad`) with CUDA 12.1, cuDNN 8.9.7 and NCCL 2.19.1, using `torchvision` 0.17.0, `timm` 1.0.28, `webdataset` 1.0.2, NumPy 1.26.4 and `zstandard` 0.25.0 on `libzstd` 1.5.7, on the hardware of Appendix B.5.

We state one limitation of this protocol explicitly. The training script sets no global random seed. The three repeated ResNet-18 runs of Section 4.2 are therefore repeats of a bit-identical command rather than a seed sweep, and the spread we quote, 0.083 points of accuracy and 0.015 of packet, measures the variation due to the unseeded data order and augmentation and non-deterministic GPU reductions. That is the noise floor a reader should compare our differences against, and we use it for that purpose throughout. It is measured on ResNet-18 only: the other networks, trained with different batch sizes, optimizers and distillation, may have a different spread, so on them it is an indicative reference rather than a bound. It is not a measurement of sensitivity to initialization, which we do not claim.

B.3 Consistency Across Architectures

Every claim in this paper is a difference between runs, so the parts of the training recipe that are not the difference under study have to be identical, and the two lists below record which parts are held fixed across the runs of Tables 1, 6, 8 and 10 and which are varied per architecture, with the reason; nothing else differs.

Identical everywhere

- quantizer LSQ, uniform grid
- step-size init by MSE grid search
- no gradient scaling on the step sizes
- no external pruning rule of any kind
- dual step 1.3×10^{-2}, relative
- $N_{\mathrm{dual}} = 3$, six on the two ViT rows with the entropy term
- dual called every 4 optimizer steps
- one warm-up epoch before the entropy term
- step-size rate follows the weight cosine
- per-tensor step sizes in every run of Tables 1, 6, 8 and 10
- $T_1 = 10^{-5}$ in every PRESTO run, and 0 in the $(0, 0, 0)$ LSQ baselines

Varied, and why

- learning rate and its floor: tuned per network
- batch size: memory, and the teacher when distilling
- epoch budget: compute, stated in every row
- distillation: on both transformers at $\alpha = 0.9$ and on AlexNet and EfficientNet-B0 at $\alpha = 0.5$, off on both ResNets
- eight bits: on the first two convolutions of AlexNet and on 26 early and depthwise tensors of EfficientNet-B0
- step-size rate: relative on AlexNet and EfficientNet-B0, absolute elsewhere
- per-channel step sizes: on the superseded runs, the EfficientNet-B0 rows of Table 2 and all of Tables 9 and 11
- weight optimizer: SGD on the convolutional networks, Adam on the transformers
- learning-rate warm-up: on AlexNet and EfficientNet-B0 only

One of these entries deserves a comment. The step-size rate is expressed as an absolute value on the ResNets and the transformers, and as a fraction of the tensor's own initial step on AlexNet and EfficientNet-B0. The two forms coincide to within 11% on AlexNet, whose mean initial step is 0.0089 against the 0.01 the absolute value was calibrated on. They differ by a factor of 3.6 on EfficientNet-B0, whose mean initial step is 0.036, and there the relative form is the correct one, because it makes the per-step displacement uniform across tensors.

The dual solver dominates the cost and its complexity is $O\big(N_{\mathrm{dual}}\,(B^2 + n_\ell C)\big)$ per tensor and per call, of which the hull construction is the $O(B^2)$ part and the per-weight sweep the rest. Appendix B.5 measures that cost per call, fits it per tensor and per weight, and explains why the $N_{\mathrm{dual}} = 6$ ViT-B/16 row cannot enter the fit.

B.4 Calibrating the Coefficients

None of the three coefficients can be derived from the network in closed form, and we tried to derive them before settling on measurement. This appendix records the procedure that does work, because it is the part of the method that a practitioner has to repeat.

Dose calibration. The dose is measured, then transported by its integral. Three analytic models for the sparsity coefficient failed in sequence on AlexNet. Each missed by a wide margin, from ten times too strong to inert by three to four orders of magnitude, because each modeled how the term moves the latent weights, while its dominant effect is on the step sizes: it inflates them and widens the interval mapped to the zero symbol, which is the cheaper way of obtaining the missing mass. What replaced the derivation is a dose ramp, a run in which T_2 and T_3 grow linearly from zero to a deliberately excessive value, read one epoch at a time so that the fraction of the ramp reached becomes the dose axis.

Dose transfer. Exposure, not the coefficient, is what transfers. A ramp row reports the exposure accumulated up to that epoch, not a coefficient held for a whole run, and reproducing that row at fixed dose requires redoing the integral. Writing $\sigma = \sum_e \mathrm{lr}(e)/\mathrm{lr}_0$ for the schedule sum of the learning rate over the run, the quantity that transfers between two runs is $T \cdot \sigma$. On EfficientNet-B0 the ramp row that reads $T_2 = 1.2 \times 10^{-6}$ at epoch six reproduces on a twenty-epoch cosine only at 3.16×10^{-7}, a factor of 3.8, and the row one epoch earlier, which reads 10^{-6}, at 2.28×10^{-7}, a factor of 4.4. We call *knee* the point of the accuracy–size frontier beyond which the accuracy spent per point of size starts to rise. The knee taken from the later row, carried to a thirty-epoch cosine whose schedule sums are 10.9 and 16.2, needs 2.1×10^{-7}, which is the dose of Table 3. The EfficientNet-B0 sparsity dose and the AlexNet entropy dose were set in this way; the other doses were set on earlier runs, by spending the accuracy headroom they measured, by scaling a reference dose, or from the regression below.

Extrapolation of the entropy coefficient. The entropy coefficient extrapolates on a log scale, until it does not. Regressing the two sixty-epoch ResNet-18 rows of Tables 1 and 6 on $\ln T_3$ gives -1.31 points of size and -0.81 points of accuracy per unit. The size prediction holds well outside the fitted range: at $T_3 = 2.6 \times 10^{-7}$, two and a half times the largest fitted dose, it predicts 2.98% against the 3.02% measured, and it does so across budgets, since the deep point was run for a hundred epochs. The accuracy prediction does not: it predicts -0.90 where we measure -1.404, because the marginal cost rises by a factor of 1.6 to 2.0, depending on the budget of the reference row, once the regime of near-zero accuracy loss is left. We report both predictions because the asymmetry between them is the useful part, in the sense that the size can be planned in advance while the accuracy has to be verified by running the configuration.

Role of each coefficient. T_1 is the perspective ridge and is held at 10^{-5} everywhere. The term ablation shows that it compresses nothing on its own, which is expected: its intended role, which we have not isolated in an ablation, is not to compress but to make the relaxation of the keep-or-drop decision tight, so that the other two act on a variable priced correctly away from the integers. T_2 can be dosed to a target sparsity read off a ramp, as on EfficientNet-B0, while T_3 is dosed to a target size, and in that case the dual step has to be relative, because otherwise the coefficient does not control the outcome at all.

B.5 Hardware and Wall-Clock Cost

All runs use four nodes, each carrying four NVIDIA A100-SXM-64GB GPUs and 32 CPU cores per task, for sixteen GPUs in total. Data are read from WebDataset shards. Table 5 reports the per-epoch training time excluding the evaluation and serialization pass, as mean and standard deviation over all epochs after the first, with the global batch of Table 3. The first epoch is listed separately because the entropy warm-up leaves the dual solver switched off in it.

Table 5: Per-epoch training time, excluding metrics, for the runs of Tables 1, 6 and 8, and the $N_{\mathrm{dual}} = 3$ ViT-B/16 run of Appendix B.5.

Run	Epochs	Time per epoch [s]	First epoch [s]
ResNet-18, LSQ only	60	108 ± 1	109
ResNet-18, ridge only	60	108 ± 0	108
ResNet-18, no entropy	60	108 ± 1	108
ResNet-18, no sparsity	60	427 ± 31	110
ResNet-18, full PRESTO	60	447 ± 33	108
ResNet-18, full PRESTO	100	444 ± 27	108
ResNet-50, LSQ only	20	109 ± 0	109
ResNet-50, full PRESTO	20	997 ± 107	157
ResNet-50, $T_3 = 10^{-7}$	20	920 ± 50	157
DeiT-Small, LSQ + distillation	25	108 ± 0	107
DeiT-Small, full PRESTO	25	421 ± 13	107
EfficientNet-B0, LSQ + distillation	30	106 ± 0	106
EfficientNet-B0, full PRESTO	30	346 ± 21	106
ViT-B/16, LSQ + distillation	20	172 ± 1	169
ViT-B/16, full PRESTO, $N_{\mathrm{dual}} = 3$	20	1173 ± 59	234
ViT-B/16, full PRESTO, $N_{\mathrm{dual}} = 6$	20	2168 ± 48	234
AlexNet, ridge only	60	108 ± 0	108
AlexNet, no entropy	60	108 ± 1	108
AlexNet, no sparsity	60	678 ± 38	108
AlexNet, full PRESTO	60	721 ± 85	108

Two costs are separable in that table. The perspective ridge and the sparsity term go through a per-step closed-form path and add no measurable cost on ResNet-18 and AlexNet: on ResNet-18 the ridge-only and no-entropy runs cost 108 seconds per epoch against 108 for plain LSQ. The entropy term is the expensive one, because it calls the dual solver once every four optimizer steps: the same network costs 447 seconds per epoch with it, a factor of 4.1. The same separation holds on AlexNet, whose ridge and sparsity arms run at 108 seconds per epoch and whose two arms with the entropy term run at 678 and 721, factors of 6.3 and 6.7.

That factor is not a property of the network size alone, and comparing networks by it requires holding N_{dual} fixed, since the two ViT-B/16 rows of Table 5 differ in exactly that. Read at $N_{\mathrm{dual}} = 3$

throughout, dividing the entropy overhead by the 313 dual calls of each epoch makes one call cost 1.08 seconds on ResNet-18, 2.84 on ResNet-50 and 3.20 on ViT-B/16. Per tensor those are 51.6, 52.5 and 64.0 milliseconds, nearly flat, whereas per million quantized weights they are 93, 111 and 37, a factor of three apart: ViT-B/16 carries 3.4 times the weights of ResNet-50 across four fewer tensors and costs only 13% more per call. The solver builds one lower hull and runs one FISTA loop per tensor, and over the whole range we measured it is dominated by that per-tensor overhead rather than by the per-weight sweep. A least-squares fit gives 48 milliseconds per tensor and 9 nanoseconds per weight, with the per-tensor term accounting for between 75% and 92% of a call. At $N_{\mathrm{dual}} = 6$ the same ViT-B/16 configuration costs 6.38 seconds per call, almost exactly twice as much, which is what the linear dependence on N_{dual} predicts and which is why the two settings cannot be put on one scaling argument. Batching the per-tensor work is the obvious optimization and we have not attempted it.

Relative overhead. Against its own LSQ baseline the entropy term costs 3.9× on DeiT-Small, 3.3× on EfficientNet-B0, 4.1× on ResNet-18, 6.8× on ViT-B/16 at $N_{\mathrm{dual}} = 3$ and 9.1× on ResNet-50. The two ViT-B/16 frontier points are the only runs with an active entropy term at $N_{\mathrm{dual}} = 6$, a value the released configurations also carry, inertly, on the ViT-B/16 baselines, which doubles the dual iterations per call, and with them the traversals of the dual interval, and raises the overhead to 12.6×. We ran the headline ViT-B/16 operating point at both settings: $N_{\mathrm{dual}} = 6$ gives 80.372% at 4.42% of the checkpoint and $N_{\mathrm{dual}} = 3$ gives 80.260% at 4.44%. The difference, 0.112 accuracy points, is 1.3 standard deviations of the spread between our three ResNet-18 repeats and favors the larger value, which is why the ViT-B/16 rows use it; but both settings clear the standardized codec at that size by more than five accuracy points, so the ViT-B/16 claim does not depend on the choice.

C Further Experimental Results

This appendix reports the experiments that support Section 4 without being needed in order to follow it. Nothing it states there in summary is left without its numbers here. The raw accuracies behind Figure 1 are in Tables 1, 6 and 8, except five points of the published NNCodec grids, taken from Becking et al. (2023). One published row is outside the panel of that figure, the NNCodec point at 4.85% on ViT-B/16, which is 5.85 accuracy points below its checkpoint; Table 1 reports it.

Table 6: Additional configurations and published points, with the size and accuracy conventions of Table 1.

Architecture	Variant	Epochs	Top-1 [%]	Size [%]	Sparsity [%]
ResNet-18	PRESTO ($10^{-5}, 10^{-7}, 10^{-7}$)	60	69.606	4.22	68.6
ResNet-18	PRESTO ($10^{-5}, 10^{-7}, 2.6{\cdot}10^{-7}$)	100	68.330	3.02	80.6
ResNet-50	PRESTO ($10^{-5}, 10^{-7}, 10^{-7}$)	20	75.444	3.73	75.3
ResNet-18	Metz et al. (2024)	–	69.0 [69.8]	3.57	–
ResNet-18	LilNetX[‡§] (Girish et al., 2023)	–	−1.2 [70.0]	2.16	–
ResNet-50	Metz et al. (2024)	–	75.6 [76.1]	4.41	–
ResNet-50	EPR-DFT[‡§] (Oktay et al., 2020)	–	−1.0 [75.0]	5.38	–
ResNet-50	LilNetX[‡§] (Girish et al., 2023)	–	−2.9 [76.6]	3.53	–
ResNet-50	DeepCABAC[§] (Wiedemann et al., 2020)	–	−0.47 [74.98]	10.14	–
ResNet-50	DeepCABAC, sparse[§] (Wiedemann et al., 2020)	–	−0.44 [74.09, pruned]	5.14	–
ViT-B/16	NNCodec (Becking et al., 2023)	–	80.41 [81.07]	10.44	–
EfficientNet-B0	NNCodec (Becking et al., 2023)	–	76.29 [77.67]	10.08	–
ViT-B/16	PRESTO ($10^{-5}, 10^{-8}, 3{\cdot}10^{-9}$), one fifth	20	80.682	7.13	55.7
DeiT-Small	PRESTO ($10^{-5}, 2.5{\cdot}10^{-8}, 7.5{\cdot}10^{-9}$), half	25	79.704	10.63	30.9
DeiT-Small	PRESTO ($10^{-5}, 10^{-8}, 3{\cdot}10^{-9}$), one fifth	25	79.694	11.99	21.5
DeiT-Small	Q-ViT 4-bit (Li et al., 2022)	300	80.9 [79.9]	12.93*	–

C.1 Code Length under Quantization-Aware Training

One measurement motivates the method before any comparison is made. Table 7 takes the LSQ baselines of ResNet-18, ResNet-50, DeiT-Small and ViT-B/16 and reads the packet of each at the first epoch against its packet at the end, together with the entropy of its non-zero symbols over the same interval. The sizes there are shares of the FP32 bits of the quantized tensors; a packet r on that basis is $rf + (1 - f)$ of the checkpoint.

Sixty epochs of four-bit quantization-aware training on ResNet-18 take the accuracy from 67.75% after the first epoch to 70.49%, 0.756 points above the checkpoint, and at the same time make the

Table 7: Packet size and non-zero symbol entropy of the LSQ baselines of ResNet-18, ResNet-50, DeiT-Small and ViT-B/16, at the first and at the last epoch. Sizes are the packet over the FP32 bits of the *quantized tensors*, and changes are computed on the unrounded sizes.

Baseline	Architecture, epochs	Packet [%], first → last	Change [%]	H(non-zero) [bits]
LSQ only	ResNet-18, 20	$8.63 \rightarrow 9.34$	$+8.2$	$2.796 \rightarrow 3.080$
LSQ only	ResNet-18, 60	$8.59 \rightarrow 9.76$	$+\mathbf{13.7}$	$2.788 \rightarrow 3.215$
LSQ only	ResNet-50, 20	$8.83 \rightarrow 9.78$	$+10.7$	$2.854 \rightarrow 3.147$
LSQ + distill.	DeiT-Small, 25	$10.84 \rightarrow 11.44$	$+5.6$	$3.355 \rightarrow 3.550$
LSQ + distill.	ViT-B/16, 20	$9.67 \rightarrow 10.19$	$+5.4$	$3.169 \rightarrow 3.316$

serialized model 13.7% larger, and the entropy of the symbol histogram rises in all of them. A low bit width therefore does not imply a low entropy, and training left to itself drifts in the wrong direction. Under the regularizer the same quantity falls, at the operating points of Table 1, by between 0.553 bits on ResNet-50 and 1.415 bits on ViT-B/16, so PRESTO does not improve an existing trend but reverses it.

C.2 Term Ablation and Mechanisms

Two ablations, one on ResNet-18 and one on AlexNet. Within each, the runs share every flag but the coefficient triple, the budget and the checkpoint included. The full-precision accuracies are 69.734% on ResNet-18 and 56.524% on AlexNet, and the LSQ baseline of each block, from which Table 8 measures the rates, is the $(0, 0, 0)$ arm on ResNet-18 and the ridge-only arm on AlexNet. One caution belongs with the exchange rates of Table 8: they are secants to a concave frontier between arms that do not end at the same size, so the arm dosed further shows the flatter secant, and we do not use them to rank the two terms.

Table 8: Term ablation on ResNet-18 and AlexNet at sixty epochs. *Rate* is points of size bought per point of accuracy spent, relative to the LSQ baseline of the block; H is the entropy of the non-zero symbols, pooled over the quantized tensors, at the end of the run.

Architecture	(T_1, T_2, T_3)	Top-1 [%]	Size [%]	Rate	H [bits]
ResNet-18	$(0, 0, 0)$	70.490	9.84	–	3.215
	$(10^{-5}, 0, 0)$	70.392	9.87	–	3.210
	$(10^{-5}, 10^{-7}, 0)$	70.312	6.97	16.15	2.526
	$(10^{-5}, 0, 6{\cdot}10^{-8})$	70.090	5.03	12.05	1.926
	full	70.020	4.89	10.54	1.901
AlexNet	$(10^{-5}, 0, 0)$	56.500	10.57	–	3.334
	$(10^{-5}, 3{\cdot}10^{-7}, 0)$	55.770	3.03	10.32	1.688
	$(10^{-5}, 0, 2.3{\cdot}10^{-8})$	55.882	3.47	11.48	1.721
	full	55.644	2.84	9.03	1.686

One asymmetry belongs with the single-term shares of Section 4.3: with $T_3 = 0$ the regularizer reaches the weights at every step, whereas with $T_3 > 0$, after the warm-up epoch, only when the dual runs, once every four, so the shares compare coefficient and frequency together. The bias runs against the entropy arms, which still reach further on ResNet-18.

The AlexNet entropy dose was not transferred. Carrying ResNet-18's 6×10^{-8} across put the arm far past its operating point: after 25 of its 60 epochs, when we stopped it, it stood at 2.54% of the checkpoint but 1.96 accuracy points below it, and the resulting exchange rate, 4.15, would have made the entropy term look far worse than it is. That run traced the dose–response curve one epoch at a time and the corrected dose, 2.3×10^{-8}, was read off it by the exposure rule of Appendix B.4. It is a concrete instance of the general point made there: an exchange rate measured at a transferred dose is not a property of the term.

Mechanisms. The two terms act through different mechanisms. On ResNet-18, measured on the matched sixty-epoch arms of Table 8, the sparsity term contracts the parameter distribution furthest: its first percentile moves 14.31% towards zero, against 11.66% for the ridge alone and 11.25% for the entropy term. The entropy term works on the assignment instead. It cuts the value bits by 65% against the sparsity term's 24%, and the entropy of the non-zero symbols by 0.858 bits, where the

Table 9: Change from the first to the last epoch in the per-channel EfficientNet-B0 runs of Table 11. "Step size" is the median over tensors of the ratio between their final and initial mean LSQ step size; "p_1" is the relative change of the first percentile of all parameters (the latent weights and the $<1\%$ kept in floating point); "entropy" is the entropy of the non-zero symbols, pooled over the quantized tensors, in bits per surviving weight; mask and values are in percentage points of the FP32 bits of the quantized tensors. Zeros are counted as in Table 11.

Run	Step size	p_1 [%]	Mask [points]	Values [points]	Entropy [bits]
LSQ baseline $(0, 0, 0)$	$\times 0.980$	-3.07	$+0.037$	-0.152	-0.036
T_2 only	$\times 1.353$	-6.99	$+1.049$	-4.558	-1.177
T_3 only	$\times 0.987$	-3.23	$+0.172$	-0.649	-0.156

sparsity term moves that entropy by only 0.024 and the LSQ baseline lets it rise by 0.427. The arms do not start from a common value, 2.502 bits under the sparsity term against 2.78 elsewhere, because the first measurement is taken at the end of the first epoch and the closed-form terms act from the first optimizer step while the entropy term is still in its warm-up, so the quantity to compare is the change along each run and not its endpoints. How each term creates its zeros shows in the step sizes, which every log records. On AlexNet the sparsity arm inflates them, at a median final-to-initial ratio of 1.78 against 0.96 for the LSQ baseline, and on EfficientNet-B0 by 35% (Table 9), widening the interval that maps to zero; on ResNet-18 it does not, at 0.65 against 0.68 for the LSQ baseline, its zeros coming instead from the contraction of the weights measured above, and there it is the entropy arm that holds the step sizes up, at 1.02. Those three runs differ in budget, ten epochs for the LSQ baseline and the entropy arm and twenty for the sparsity arm, and in dose, so the table documents where each term acts and not how strongly; the sparsity arm in particular, at nineteen times the sparsity coefficient of the EfficientNet-B0 run of Table 1, also lowers the entropy of the non-zero symbols.

The separation is therefore not pruning against coding: both arms create zeros (Section 4.3), and on ResNet-18 only the entropy term also shortens the code of the surviving weights, which Section 2.2 explains we report as a measurement.

Interaction of the two terms. The two terms do not add. Whichever term arrives first, the other has little left to remove, and both ablations show it on the same quantity. On AlexNet the two arms reach the same symbol distribution by different routes: the entropy of the non-zero symbols starts at 2.892 bits under the sparsity term and 3.326 under the entropy term, and *both drive it to the same floor*, 1.688 and 1.721 bits, with the combination ending at 1.686, which is indistinguishable from sparsity alone. On ResNet-18 the floor is not shared, because the sparsity term stops at 2.526 bits while the entropy term reaches 1.926, and there the combination ends at 1.901, which is indistinguishable from entropy alone. In both blocks the full objective lands on the floor of whichever single term went lower, and the second term then contributes the few percent of size reported in Section 4.3 rather than a second reduction. Which term gets there first is the architecture-dependent quantity, and it is the same ordering as the shares in Section 4.3.

C.3 Budget and Dose on ResNet-18

Beyond the comparison of Section 4.2, the hundred-epoch run at $T_3 = 10^{-7}$ was still gaining 0.05 points per epoch over its last ten when the schedule ended. Raising or lowering T_3 trades one axis against the other, whereas lengthening the schedule does not. The frontier does bend, and we report where: the marginal cost per point of size is 0.621 accuracy points between the sixty-epoch rows of Tables 1 and 6 and 1.249 between the two hundred-epoch ones, the deeper of which is at 3.02% and 1.404 below.

C.4 Joint versus Sequential Optimization

PRESTO optimizes sparsity and code length together, whereas the obvious alternative is to optimize them in sequence as the classical pipeline does, and the question is whether the joint formulation earns the complexity that it adds. We answer that question by decomposing our own objective rather than by comparing against a different method, so that nothing except the ordering changes.

Two pipelines run twenty epochs on one term and then twenty on the other, in both orders, starting from the same checkpoint and using the same coefficients as the joint run. Each stage is a complete twenty-epoch cosine, identical in shape to the joint reference, so the schedules are comparable stage by stage. The second stage reloads the latent weights and re-initializes the step sizes by MSE, which is what a sequential method does and is also, as we will see, part of the mechanism. The main confound that remains is the budget, and it favors the pipeline, which receives forty epochs against the twenty of the joint run, so we also report the joint run at forty epochs, where the two are budget-matched.

Table 10: Joint objective against the same objective split into two sequential stages, ResNet-18, $(T_1, T_2, T_3) = (10^{-5}, 10^{-7}, 3 \cdot 10^{-8})$ throughout.

Optimization	Epochs	Top-1 [%]	Size [%]
Joint	20	69.668	5.83
Joint	**40**	**70.080**	**5.82**
Sparsity alone, first stage	20	70.098	7.47
Sparsity then entropy	$20 + 20$	70.114	6.07
Entropy alone, first stage	20	69.782	6.08
Entropy then sparsity	$20 + 20$	70.108	7.21

At matched budget the three accuracies agree to within 0.034 points, well inside the 0.083 spread between the three sixty-epoch ResNet-18 repeats, so the comparison is purely on size. At forty epochs the joint run is 4.2% smaller than the better pipeline and 19.3% smaller than the worse one, and the sharper way of reading Table 10 is to ask what the second twenty epochs produce. Against the joint run at twenty epochs, which is the common reference for all three, every configuration ends between 0.41 and 0.45 accuracy points higher, while the joint run holds its size to within 0.01 points of size, and the two pipelines end 0.25 and 1.39 points of size above it. Read instead against each pipeline's own first stage, which the table also reports, the asymmetry is starker: the entropy stage of the first pipeline removes 1.39 points of size in its twenty epochs while the accuracy moves by $+0.016$, whereas the sparsity stage of the second adds back 1.13 points of size while the accuracy rises by 0.326. Sequential optimization therefore spends the extra budget on accuracy that it can no longer spend on size.

The mechanism behind that behavior is visible in the entropy of the non-zero symbols. Re-quantizing between the two stages costs 0.343 bits in one order and 0.445 bits in the other (measured at the end of the first epoch of the second stage), because the grid is re-fitted to the latents that were loaded and the symbols are remapped. The entropy stage recovers that cost and more, at -0.742 bits over its twenty epochs, whereas the sparsity stage cannot recover it and adds a further $+0.040$. Running the entropy term first therefore loses most of what it gains: it lowers the entropy by 0.759 bits, the re-fit gives back 0.445 of them and the sparsity stage adds 0.040, so the pipeline ends at 2.516 bits, 0.274 below where it began but 0.485 above where the entropy stage had brought it, against 2.112 bits for the opposite order and 2.034 for the joint run at the same budget. Ordering matters, in the direction that the classical pipeline happens to get right, and both orders lose to solving the two problems together, the re-fit they carry included.

C.5 Accuracy–Size Frontier on Vision Transformers

On DeiT-Small and ViT-B/16 the LSQ baseline stays below the accuracy of the full-precision checkpoint, whereas the ResNet LSQ baselines rise above it. We therefore do not claim losslessness in accuracy on these two networks: we report instead the accuracy–size trade-off against an LSQ baseline trained for as many epochs as the PRESTO runs. On ViT-B/16 five runs without the regularizer, spanning learning rates from 3×10^{-5} to 10^{-4} and budgets from ten to thirty epochs land within a tenth of a point of each other and about two tenths below full precision, from -0.142 to -0.242: there is no headroom to spend, and budgets up to thirty epochs create none. Those five predate the packed-attention fix of Appendix B.2, so only their accuracies are used here and their sizes are not reported.

Against that LSQ baseline PRESTO removes 58% of the ViT-B/16 packet for 0.444 accuracy points, reaching 4.42% of the checkpoint at 72.9% sparsity, the highest of any transformer run here. On DeiT-Small the curve is traced by the LSQ baseline and three doses, obtained by scaling both co-

efficients by a common factor, and the marginal cost rises monotonically: 0.001, 0.208 and 0.325 accuracy points per point of packet, measured from the LSQ baseline through the half, reference and double doses. The first of those is the interesting one, because between the LSQ baseline and half the dose the packet falls from 12.00% to 10.63% while the accuracy moves by 0.002 points, which is far inside the run-to-run spread we measure on ResNet-18, so on this network the first 11% of compression is free and the trade-off only begins beyond that point.

A fourth dose, one fifth of the reference, reported in Table 6, deserves a separate mention, because there the sparsity rises from 13.7% to 21.5% and the packet barely moves, from 12.00% to 11.99%. Below 50% sparsity the support mask grows with every new zero, here by 0.49 points of the checkpoint, and the value stream, which at the LSQ baseline's cost per non-zero symbol would have shrunk by 0.87 points, shrinks by only 0.50 because the entropy of the non-zero symbols rises from 3.55 to 3.71 bits. Past $p = 1/2$ the mask shrinks as well, which is why our most compressed ResNet-18 rows carry a cheaper mask than less sparse ones.

Other published points on DeiT-Small. The four-bit row of Q-ViT (Li et al., 2022) is stronger on accuracy than ours but sits at a nominal 12.93%, larger than our own LSQ baseline, and comes after three hundred epochs, so it matches neither of our two axes. FTerViT (Ruciński et al., 2026) ternarizes to a bit-packed 6.58% at 77.47%, where our 7.74% row reaches 78.948%. The Q-ViT four-bit row is in Table 6 and the FTerViT row in Table 1.

C.6 Analysis of the AlexNet Deficit

Section 4.4 reports that published prune-first methods remain ahead of us on AlexNet, a network whose parameters sit overwhelmingly in three dense layers, where a pipeline of magnitude pruning, iterative retraining, k-means weight sharing and Huffman coding is at its strongest. Two explanations can be ruled out. It is not the budget, since the full objective has converged at sixty epochs, its last ten changing the accuracy by $+0.001$ points per epoch, against $+0.010$ on the LSQ baseline, -0.009 on the entropy arm and $+0.022$ on the sparsity arm. It is also not a retraining phase too short to recover from the pruning, which is the natural suspicion given how Deep Compression is trained. In the sparsity arm and the full objective the sparse structure is 95% final by epochs 20 and 17 and leaves twenty-nine to thirty-two epochs above a tenth of the base rate, against three on ResNet-18, where it settles only at epoch 46. AlexNet therefore gets an order of magnitude more high-rate retraining on its final structure than the network that finishes above full precision, and the full objective still ends 0.880 accuracy points below the checkpoint.

What we believe it is instead is allocation, and the arithmetic is worth writing down even though we did not run a controlled arm that would test it. In every run of Tables 1 and 8 our sparsity is not allocated: one coefficient drives all eight tensors, whose sizes span three orders of magnitude, while both methods ahead of us choose a sparsity level per layer. The five convolutions hold 4.04% of the weights, so sparing them, at the 3.38 bits per weight of the AlexNet LSQ baseline instead of the 0.90 of the full objective, would cost about 0.3 points of packet. That is worth 0.03 accuracy points at the exchange rate we measure here, and about three extra points of sparsity on the dense layers would repay it: we spend the feature extractor to save a third of a point that the dense layers would return for almost nothing. This is a hypothesis and not a measurement, and a controlled per-layer dose at the recipe of this paper is left to future work. One earlier run did carry a per-layer dose, and we say below why it does not settle the question.

Earlier runs on AlexNet and EfficientNet-B0. Table 11 collects earlier per-channel runs on these two networks, with sizes on the whole-checkpoint accounting of Table 1; its LSQ baseline rows carry no sparsity and no entropy term, and the AlexNet one keeps the ridge at 10^{-5}. On AlexNet, sweeping T_2 traces a dose–response curve; on the training recipe of this paper the exchange rate is still 10.3 points of size per accuracy point at 78% sparsity, against a knee between 42% and 51% on the earlier per-channel training recipe, so the knee is a property of the procedure and not of the network. Section 4.4 reports where we stand there and why. EfficientNet-B0 is the opposite case, with its knee at 23%: a network designed for accuracy per parameter has less redundancy to remove.

That AlexNet row is the one run of this paper whose sparsity is allocated, and it is the run alluded to in the hypothesis above, so we state what it is and what it is not. Its T_2 is a vector of eight per-layer values, calibrated by a displacement rule, the T_2 of each tensor being the value whose pull,

integrated over the schedule and ignoring the task gradient, would carry a weight at that tensor's target quantile of $|w|$ into the zero cell of the quantizer, with one common factor setting the overall dose, from per-layer target sparsities of 0.112, 0.434, 0.455, 0.441 and 0.441 on the five convolutions and 0.637, 0.637 and 0.525 on the three dense layers, and read along the dose ramp rather than held at a fixed dose. The allocation therefore points the way the hypothesis above asks for, sparing the feature extractor and spending the dense layers, and it did not pay: this row ends at 70.60% sparsity and 3.67% of the checkpoint for 1.586 accuracy points, against 78.20% and 3.03% for 0.754 points on the single-coefficient arm of Table 8. We do not read that as evidence against the hypothesis, because the two runs differ in four other ways: per-channel rather than per-tensor step sizes, which Appendix C.7 prices on its own; a forty-epoch dose ramp rather than a fixed dose; a learning-rate floor five times higher; and an endpoint 7.6 points of sparsity lower, so the two are not read at the same place on a concave frontier. We also logged the per-layer coefficients but not the per-layer sparsities they produced, so we cannot check that the targets were met. The controlled arm, one dose per layer at the per-tensor recipe of this paper with everything else held fixed, is the one we did not run.

Table 11: Earlier per-channel runs on AlexNet and EfficientNet-B0. The per-channel runs of Tables 2, 9 and 11 count as zero every quantized value of magnitude at most 10^{-3}; this affects their sizes and sparsities but not their accuracies, and all per-tensor results are unaffected. †the one run of this paper that allocates its sparsity coefficient per layer, discussed in the text; every other run drives all of its tensors with one coefficient.

Architecture	Variant	Epochs	Top-1 [%]	Δ FP32 [points]	Size [%]	Sparsity [%]
AlexNet	LSQ + ridge + distillation	60	56.552	$+0.028$	10.77	16.11
AlexNet	PRESTO, T_2 only†	60	54.938	-1.586	3.67	70.60
EfficientNet-B0	LSQ + distillation	10	76.590	-0.976	12.42	16.19
EfficientNet-B0	PRESTO, T_3 only	10	76.452	-1.114	12.07	18.06
EfficientNet-B0	PRESTO, T_2 only	20	74.746	-2.820	9.06	39.83

C.7 Bit Accounting

Support mask. The percentages in this subsection are shares of the FP32 bits of the quantized tensors, the accounting in which the packet decomposes additively; over the whole checkpoint each share is multiplied by f, and the residue $1 - f$ is added once to the packet. The support mask is compressed with the same coder as everything else, and the i.i.d. Bernoulli model $H(p)/32$ of FP32 storage is a first-order description of its cost rather than the cost itself: at 51.53% sparsity the model gives 3.12% and the serializer produces 2.905%, because the mask is not memoryless. What the model does capture is the shape, since it is maximized at $p = 0.5$. Past that point extra zeros are rewarded twice, by a shorter value stream and by a cheaper mask, and our runs show it: at 51.53% sparsity the mask costs 2.905%, at 67.37% it costs 2.608% and at 68.55% it costs 2.563%. A method that stops at half sparsity sits exactly on the worst point of that curve.

Per-channel grids. Per-channel grids are paid for twice. Step sizes per output channel are usually treated as accuracy that comes for free. On EfficientNet-B0, switching the LSQ baseline from per-channel to per-tensor step sizes, with everything else unchanged (both runs give eight bits to seventeen tensors, the input convolution and the sixteen depthwise ones), cost 0.038 accuracy points and returned 1.04 points of packet, a far more favorable rate than the one the sparsity term gives at its knee. The per-channel sizes here count zeros as in Table 11. Of that return, 0.597 was the metadata, a hard floor of one FP32 number per output channel over 82 tensors, and 0.094 the support mask. The remaining 0.345 came from the value stream, because one shared grid per tensor gives the coder a single symbol distribution instead of one distribution per output channel and the pooled stream is more concentrated. Per-channel quantization therefore costs code length as well as metadata, and on networks with many small tensors the two together are worth more than the accuracy that it provides.

C.8 Learning and Compression Curves

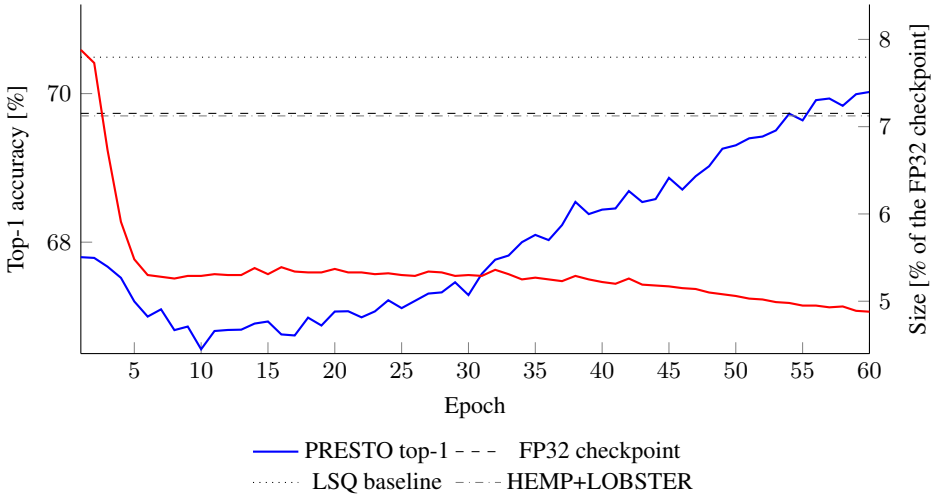

Figure 3: Top-1 accuracy (blue, left axis) and packet as a percentage of the checkpoint (red, right axis) over sixty epochs for the headline ResNet-18 configuration; the horizontal reference lines are accuracies, read on the left axis, and HEMP+LOBSTER, at 69.70%, lies within 0.034 points of the FP32 checkpoint, so the two are not separable at this scale. The size curve is the only one on the right axis and carries no legend entry.

The two phases of a run are visible in Figure 3, which follows one of the three repeats of the headline configuration $(T_1, T_2, T_3) = (10^{-5}, 10^{-7}, 6 \times 10^{-8})$ down to 4.88% of the checkpoint. The compression is delivered early, within the first ten epochs, while the accuracy recovers over the whole cosine and crosses the full-precision checkpoint only near its end.

C.9 Negative Results and Cost

Four failures shaped the method and we report them because each changed what we believed; a fifth entry is not a failure but a measured cost, and it bounds what the method is worth.

Dual convergence. The entropy coefficient is only as strong as its dual is converged. Raising T_3 tenfold once left the applied gradient within 0.013% of itself and the model compressed no more than its LSQ baseline, because the coefficient reaches the weights through the multiplier alone and an absolute dual step does not scale with it; Appendix A.4.2 derives the rate and reports the measurement. The general lesson is that when a coefficient looks inert, the dual variable through which it acts should be checked for convergence before the coefficient itself is touched.

Dose calibration. The dose cannot be derived, only measured and transported. Three analytic models for the sparsity coefficient failed in sequence, each for a different reason, and Appendix B.4 reports them together with the procedure that replaced them. The general lesson is that what transfers between runs is the cumulative exposure, the coefficient times the schedule sum, and never the coefficient itself.

Stopping criterion. A stopping criterion we wrote would have given the wrong answer. We had committed to abandoning the entropy channel on EfficientNet-B0 if it did not separate from the LSQ baseline within four epochs even at a much larger entropy coefficient. It did not separate, but the cause was the dual rather than the dose, and the channel opened as soon as the step was made relative.

Latent-weight accuracy. One diagnostic is worthless and we stopped using it. The accuracy of the network evaluated at its latent weights collapses by tens of points without predicting anything about the deployed model, because the straight-through estimator applies a gradient computed at the quantized point to a latent that has no restoring force toward it. On the networks with batch normalization this evaluation also uses statistics recalibrated for the quantized weights, which by itself accounts for much of the collapse. It does not appear in this paper.

Training cost. The regularizer is expensive, and the cost is not intrinsic. The entropy term costs $3.3\times$ to $12.6\times$ the training time of its LSQ baseline, while the sparsity term needs no solver and almost suffices on AlexNet (Section 4.3, Appendix B.5); the solver is dominated by a per-tensor overhead that batching would remove, which we have not attempted. On DeiT-Small the total still favors us, at about ninety-five baseline-equivalent epochs against the three hundred that Q-ViT reports.

D Glossary of Notation and Acronyms

Table 12 collects the mathematical objects and the principal acronyms used throughout the paper. Superscript (ℓ) identifies a quantized tensor, subscript i identifies a weight within that tensor, subscript b identifies a non-zero codebook symbol, superscript $*$ denotes a selected optimal value, and superscript t or subscript k denotes an algorithmic iteration as specified below.

Table 12: Glossary of the notation and acronyms used in the paper.

Symbol or acronym	Meaning
Data, network, and indexing	
θ	Collection of all network parameters, partitioned as $(w, \theta_{\mathrm{fp}})$.
w	Collection of latent weight tensors selected for quantization.
θ_{fp}	Biases, normalization parameters, and other parameters retained in floating-point precision.
Λ	Index set of the quantized weight tensors.
ℓ	Index of a quantized tensor, with $\ell \in \Lambda$.
$w^{(\ell)}$	Latent weight vector of tensor ℓ.
n_ℓ	Number of scalar weights in tensor ℓ.
N	Number of parameters of the whole network, $\lvert\theta\rvert$.
$\mathcal{I}_\ell$	Weight-index set $\{1, \ldots, n_\ell\}$.
i	Index of a scalar weight within tensor ℓ.
$\mathcal{L}_{\mathrm{task}}$	Empirical task loss over the training set, including the distillation term where it is used.
$\mathcal{R}_{\mathrm{PRESTO}}$	Complete PRESTO regularizer $\sum_{\ell\in\Lambda} \Phi_\ell$.
$\mathcal{J}$	Complete outer training objective.
Reporting and calibration	
f	Fraction of a checkpoint's parameters that are quantized.
α	Weight of the distillation term in the training loss.
κ	Common factor by which a dose scales T_2 and T_3.
r	Packet size as a fraction of the FP32 bits of the quantized tensors; distinct from the normalized weight $r_i^{(\ell)}$.
p	Sparsity, the fraction of weights assigned to the zero symbol.
σ	Schedule sum of the learning rate over a run (Appendix B.4).
m	Non-zero mass $\sum_{b\in\mathcal{B}} c_b^{(\ell)}$ of a tensor (Appendix A.6).
π_b	Conditional distribution $c_b^{(\ell)}/m$ over the non-zero symbols.
$H(\cdot)$	Shannon entropy in bits.
Learned-step quantization (LSQ)	
Q_n, Q_p	Minimum and maximum integer quantization codes.
$\mathcal{Q}$	Integer codebook $\{q \in \mathbb{Z} : Q_n \le q \le Q_p\}$.
q	Generic integer code in $\mathcal{Q}$.
C	Total number of symbols, $C = \lvert\mathcal{Q}\rvert$.
a_ℓ	Learned quantization step size of tensor ℓ.
a	Collection $\{a_\ell\}_{\ell\in\Lambda}$ of all step sizes.
$r_i^{(\ell)}$	Normalized latent weight $w_i^{(\ell)}/a_\ell$.
$\bar{w}_i^{(\ell)}(a_\ell)$	Latent weight clipped to the representable interval of tensor ℓ.
$\bar{w}^{(\ell)}$	Vector of clipped latent weights in tensor ℓ.
$\widehat{w}_i^{(\ell)}$	Quantized weight used in the forward pass.

Continued on the next page

Table 12 continued

Symbol or acronym	Meaning
$\widehat{w}$	Collection of all quantized weight tensors.
PRESTO inner problem	
$\mathcal{B}$	Set $\mathcal{Q} \setminus \{0\}$ of non-zero symbols.
B	Number of non-zero symbols, $B = \lvert\mathcal{B}\rvert = C - 1$.
b	Index of a non-zero symbol in $\mathcal{B}$.
q_b	Integer code associated with non-zero symbol b.
$v_b^{(\ell)}(a_\ell)$	Real-valued level $a_\ell q_b$ of symbol b in tensor ℓ.
$v^{(\ell)}, v$	Vector of real-valued non-zero levels, with the tensor superscript suppressed locally when unambiguous.
$x_{i,b}^{(\ell)}$	Nonnegative mass assigning weight i of tensor ℓ to non-zero symbol b.
$x_i^{(\ell)}, x^{(\ell)}$	Assignment masses for one weight and for the complete tensor, respectively.
$y_i^{(\ell)}$	Total mass representing weight i through non-zero symbols.
$z_i^{(\ell)}$	Mass $1 - y_i^{(\ell)}$ assigned to the zero symbol; distinct from the temporary dual-update vector $z_b^{(\ell,t+1)}$.
$c_b^{(\ell)}$	Relaxed count of non-zero symbol b in tensor ℓ.
T_1	Coefficient of the perspective ridge term.
T_2	Coefficient penalizing mass assigned to non-zero symbols.
T_3	Coefficient of the entropy term.
$\Phi_\ell(\bar{w}^{(\ell)}, a_\ell)$	Optimal value of the tensor-level PRESTO inner problem.
Lagrangian relaxation and decomposition	
$\xi_b^{(\ell)}$	Unrestricted Lagrange multiplier of the count-defining constraint for symbol b.
$\xi^{(\ell)}$	Vector in $\mathbb{R}^B$ collecting the non-zero-symbol count multipliers of tensor ℓ.
$\mathscr{L}_\ell$	Partial Lagrangian obtained by dualizing the count constraints $c_b^{(\ell)} = \sum_{i \in \mathcal{I}_\ell} x_{i,b}^{(\ell)}$ for every $b \in \mathcal{B}$.
$\varphi_{\bar{w}^{(\ell)}, a_\ell}^{(\ell)}$	Concave Lagrangian dual function obtained by minimizing $\mathscr{L}_\ell$ over the retained primal constraints.
$\mathcal{F}_i^{(\ell)}$	Feasible set of the per-weight assignment and mass variables.
$\mathscr{L}_\ell^{\text{full}}$	Inner Lagrangian including both the count and representation multipliers.
Per-weight solution	
$K_i(y_i; \bar{w}_i, \xi, v)$	Optimal value of the linear assignment subproblem for weight i at fixed non-zero mass y_i.
$u_{i,b}$	Normalized assignment mass $x_{i,b}/y_i$; distinct from the generic scalar u in the perspective definition.
$\widetilde{w}_i$	Rescaled target $\bar{w}_i/y_i$.
P_b	Point (v_b, ξ_b) used to construct the lower convex hull.
$E_{\text{l}}, E_{\text{r}}$	Leftmost and rightmost vertices of the lower convex hull.
b_-, b_+	Indices of the endpoints of the active lower-hull facet.
s_i	Intercept of the active lower-hull facet; distinct from the dual residual $s_b^{(\ell)}$.
$V_i^{\max}$	Largest representable magnitude having the same sign as $\bar{w}_i$.
$y_i^{\min}$	Smallest non-zero mass capable of representing $\bar{w}_i$.
$G_i(y_i)$	Scalar per-weight objective after eliminating the assignment variables.
y_i^{cand}	Interior stationary candidate for minimizing G_i.
y_i^*	Selected minimizer of the scalar per-weight problem.
z_i^*	Optimal zero-symbol mass $1 - y_i^*$.
G_i^*	Optimized per-weight value obtained after minimizing G_i over y_i.
$e, v_e^{(\ell)}$	Index and real-valued level of an active extreme codebook symbol.
Sensitivity and training gradients	
$\mu_i^{(\ell)}, \mu_i$	Multiplier of the representation constraint for weight i in tensor ℓ, equal to the slope of the active lower-hull facet.

Continued on the next page

Table 12 continued

Symbol or acronym	Meaning
$\beta_i^{(\ell)*}$	Selected PRESTO subgradient with respect to the clipped latent weight $\bar{w}_i^{(\ell)}$ at fixed a_ℓ.
$\delta_i^{(\ell)}$	Upstream task-loss gradient $\partial \mathcal{L}_{\text{task}} / \partial \widehat{w}_i^{(\ell)}$.
$h(r)$	Piecewise LSQ surrogate derivative of a quantized weight with respect to its step size.
$\nabla_{w_i^{(\ell)}} \mathcal{L}_{\text{task}}, \nabla_{a_\ell} \mathcal{L}_{\text{task}}$	Task-loss components with respect to a latent weight and a step size.
∇_{PRESTO}	Direction used for training: the element of $\partial \mathcal{R}_{\text{PRESTO}}$ selected by the procedure at a dual optimum, and an inexact proxy for it otherwise.
$(\nabla_{\text{PRESTO}})_{w_i^{(\ell)}}$	Latent-weight component of the selected PRESTO subgradient.
$(\nabla_{\text{PRESTO}})_{a_\ell}$	Step-size component of the selected PRESTO subgradient.
$\nabla_{\text{train}} \mathcal{J}$	Selected (sub)gradient of the complete objective passed to the network optimizers.
Dual solver and optimization order	
$\widehat{c}_b^{(\ell)}$	Count reconstructed by summing the optimal per-weight assignment masses.
$c_{b,\text{unc}}^{(\ell)}$	Unconstrained stationary count returned by the exponential relation.
$c_b^{(\ell)*}$	Count used by the dual solver after clipping $c_{b,\text{unc}}^{(\ell)}$.
$c_{\min}$	Numerical lower bound imposed on the count solution.
$c_{\max}$	Upper clamp imposed on the count solution; $c_{\max} = N$ in every run of this paper.
$s_b^{(\ell)}$	Unnormalized residual $\widehat{c}_b^{(\ell)} - c_b^{(\ell)*}$ of the relaxed count constraint; distinct from the facet intercept s_i.
N_{dual}	Number of dual iterations performed per solver call.
t	Iteration index of the dual solver, $t \in \{0, \ldots, N_{\text{dual}} - 1\}$.
η_ξ	Dual supergradient-ascent step size.
S	Number of dual calls per epoch.
$\bar{g}$	Nominal magnitude of the normalized dual residual, about $1/C$, that of a tensor holding all N parameters (Appendix A.4.2).
$g_b^{(\ell,t)}$	Dual supergradient at iteration t, normalized by $c_{\max}$.
τ_t	FISTA-style momentum parameter.
$\widetilde{\xi}^{(\ell,t)}$	Momentum-extrapolated multiplier vector.
$z_b^{(\ell,t+1)}$	Unclipped multiplier update before componentwise clipping; distinct from the zero-symbol mass $z_i^{(\ell)}$.
$\xi_{\min}^{(\ell)}, \xi_{\max}^{(\ell)}$	Lower and upper numerical bounds imposed on the dual multipliers.
Ξ_ℓ	Box $[\xi_{\min}^{(\ell)}, \xi_{\max}^{(\ell)}]^B$ onto which the multiplier is projected; the ascent runs on the dual restricted to it.
$\Phi_\ell^\square, \varphi_\ell^\square$	Inner problem and partial dual with the counts confined to $[c_{\min}, c_{\max}]$, which is what the solver poses.
ε	Floor on every assignment mass in the feasibility margin of the boxed problem.
k	Iteration index of the outer network optimizer.
$w_k, a_k, \theta_{\text{fp},k}$	Weight, step-size, and floating-point parameter collections at the start of outer step k.
$v_k, \bar{w}_k$	Collections of real-valued levels and clipped latent weights at outer step k.
$x_k^*, y_k^*, \mu_k, \beta_k^*$	Per-weight solutions and sensitivity quantities used at outer step k.
Appendix-only KKT notation	
$\rho_i^{(\ell)}$	Multiplier of the per-weight mass constraint in the boundary KKT derivation.
$\gamma_{i,b}^{(\ell)}$	Nonnegative multiplier of the constraint $-x_{i,b}^{(\ell)} \le 0$.
$\mathscr{L}_{i,\ell}^{\text{KKT}}$	Per-weight KKT Lagrangian used in the boundary derivation.

Continued on the next page

Table 12 continued

Symbol or acronym	Meaning
Principal acronyms and reporting terms	
PRESTO	Perspective-Based Entropy–Sparsity Training Optimization.
Lossless	Applied to coding, it means that the symbols are recovered exactly; the coder we use is lossless in this sense throughout. Preservation of accuracy is never implied by this word and is always stated as a number of accuracy points against a named reference.
LSQ baseline	Our own training of a network with the regularizer switched off, on AlexNet with only the inert ridge $T_1 = 10^{-5}$, for as many epochs as the PRESTO run it is compared with, unless stated otherwise (Table 3).
Packet	The complete serialized artifact: the support mask and the value stream of non-zero symbols, both compressed with `zstd`, plus the metadata, charged at 32 bits per stored step size. Sizes are reported as a percentage of the FP32 checkpoint.
Checkpoint	The public full-precision model a run starts from. It is the reference for both axes: sizes are reported as a percentage of its FP32 bits, and accuracies are compared against its own top-1 as we measure it.
Training recipe	Everything held fixed between a run and its LSQ baseline: schedule, optimizers and learning rates, epoch budget unless stated otherwise, batch size, distillation and batch-normalization recalibration. These are choices of implementation and not terms of the regularizer.
Arm	One run of an ablation, identical to the others of its block except for the coefficient triple (T_1, T_2, T_3) and, where stated, the budget.
Dose	How strongly the regularizer is applied, at a fixed ratio between its terms; the doses of a network are obtained by scaling its coefficients by a common factor. A *reference dose* is the setting from which the other doses of a network are scaled.
Frontier	The accuracy–size trade-off traced by a network as the dose is varied; its *points* are the runs that lie on it.
Knee	The point of a frontier beyond which the accuracy spent per point of size starts to rise.
Headline	The configuration of a network in the first bold row of its block in Table 1.
QAT	Quantization-aware training.
LSQ	Learned Step Size Quantization.
STE	Straight-through estimator.
FISTA	Fast Iterative Shrinkage–Thresholding Algorithm; the implementation uses a FISTA-style dual update.
KKT	Karush–Kuhn–Tucker optimality conditions.
FP32	32-bit floating-point representation.
`zstd`	Zstandard lossless compressor.
Δ	Difference between quantized and FP32 top-1 accuracy, measured in accuracy points.