Characterizing Learning in Deep Neural Networks using Tractable Algorithmic Complexity Analysis

Pedram Bakhtiarifard[1] **Sophia N. Wilson**[1] **Mahmoud H. A. Afifi**[2]
Jonathan Wenshøj[1] **Raghavendra Selvan**[1]

[1]University of Copenhagen
{pba,sophia.wilson,jowe,raghav}@di.ku.dk

[2]The American University in Cairo
mahmoudhossam@aucegypt.edu

Abstract

Training large-scale deep neural networks (DNNs) is resource-intensive, making model compression a practical necessity. The widely accepted "learning as compression" hypothesis posits that training induces structure in network weights, which enables compression. Measuring this structure through Kolmogorov-Chaitin-Solomonoff (KCS) complexity is appealing, but existing estimators based on the Coding Theorem Method (CTM) and the Block Decomposition Method (BDM) are limited to small binary objects and do not scale to modern DNNs. We introduce the Quantized Block Decomposition method (QuBD), which extends algorithmic complexity estimation to any k-ary object. QuBD first quantizes the network weights to a finite alphabet, then estimates the KCS complexity by aggregating per bit-plane CTM estimates. We show theoretically that QuBD yields a tighter upper-bound on the estimation gap for ideal prefix KCS complexity under bit-plane independence than binarization-based methods. Using QuBD, we study how the algorithmic complexity of neural network weights evolves during training, showing that it decreases as models learn, scales with data budget, increases during overfitting, follows the delayed generalization observed during grokking, and correlates with generalization performance. We further show that algorithmic information resides predominantly in the most significant bit-planes, which can serve as a practical diagnostic for determining appropriate post-training quantization levels. This work offers novel insights into learning mechanisms in DNNs by providing the first scalable, tractable estimates of KCS complexity for large, non-binary objects such as DNN weights.[1]

1 Introduction

Training recent large-scale deep neural networks (DNNs) requires energy-intensive data centers that incur a large environmental footprint (Strubell et al., 2019; Anthony et al., 2020; Sevilla et al., 2022). One way to address this is through the standard practice of compressing trained networks by removing the majority of their parameters (pruning), transferring their capabilities into smaller networks (knowledge distillation), or reducing the number of bits used to store each parameter (quantization) (Bartoldson et al., 2023). In most cases, such model compression techniques are able to drastically reduce memory, inference speed, and energy costs with only small drops in performance (Gong et al., 2014; Hinton et al., 2014; Han et al., 2016; Nagel et al., 2021). Despite these benefits, com-

[1]Code is available at https://github.com/saintslab/QuBD.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

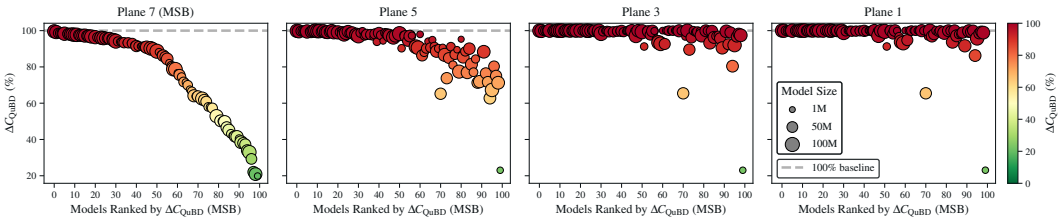


Figure 1: Relative algorithmic complexity expressed as the ratio between the complexities of pre-trained and random weights, $\Delta\mathcal{C}_{\text{QuBD}}$ (%), estimated using QuBD. We show the trend for 100 pre-trained `timm` models (Wightman, 2019) with up to 100M parameters using 8 bit-planes. Models are ordered by decreasing $\Delta\mathcal{C}_{\text{QuBD}}$ for the most significant bit-plane (MSB). Further details can be found in Sec. 4, Appx. E.1, and Appx. F.1.

pression is typically introduced as a separate stage after training, rather than as a property arising from the training dynamics themselves.

The modus operandi in DNN compression methods can be characterized as: *start with bigger models to end with smaller ones*. One widely accepted hypothesis of "learning as compression" by Shwartz-Ziv and Tishby (2017) seeks to explain the learning dynamics of DNNs that make such compression possible. Three lines of evidence support this view. First, using the information bottleneck method, Tishby et al. (2000) shows that a large part of training is devoted to learning more compact representations of the input, rather than improving performance directly. This suggests that the learned function depends on fewer distinct patterns, making it simpler to represent. Second, Valle-Perez et al. (2019) finds that training biases the networks towards simpler functions. Third, the lottery ticket hypothesis (Frankle and Carbin, 2019) demonstrates that such simple functions can be realized by small sub-networks within the original model. However, all three accounts are mostly understood in the context of feedforward networks, and describe *what* structure emerges (simpler representations and sparse sub-networks), rather than *when* it emerges or *how much* compression occurs.

Recent work by Sakabe et al. (2026) attempts to address these gaps by characterizing learning in binary neural networks through the lens of algorithmic information theory (Chaitin, 1977). They show that algorithmic complexity, estimated as structure in weights or the lack thereof, correlates strongly with learning dynamics. This provides direct empirical support for the "learning as compression" hypothesis: training induces structure in the model parameters, and this structure is precisely what enables compression. Building on this, Bakhtiarifard et al. (2026) shows that such a structure can be actively enforced throughout training without performance loss, yielding model classes that are inherently more compressible.

Despite this promise, measuring algorithmic (Kolmogorov, 1965; Chaitin, 1966; Solomonoff, 1960) (KCS) complexity is non-computable by definition and is only upper-semicomputable. Current KCS complexity estimators rely on the Coding Theorem Method (CTM), which estimates the output frequency of small Turing machines through large-scale simulations (Soler-Toscano et al., 2014). These simulations are limited to a small number of states and binary symbols, making the method practical only for small binary objects. The Block Decomposition Method (BDM) proposed by Zenil et al. (2018) scales this approach by estimating complexity from small sub-structures within larger objects. However, as we show through both theoretical and empirical analyses (Sec. 3), BDM does not scale to DNNs, thereby preventing a comparison of learning dynamics across model sizes and architectures.

To address this, we introduce the Quantized Block Decomposition method (QuBD), which quantizes any non-binary object into a finite set of integer values and estimates its KCS complexity through a bit-plane decomposition. This extends CTM-based estimators for floating-point objects like the weights of DNNs. We show theoretically that QuBD yields a tighter upper bound for estimating the ideal prefix KCS complexity of complex objects under bit-plane independence. Using QuBD, we provide the first scalable empirical characterization of the "learning as compression" hypothesis for DNNs beyond simple feedforward networks. We show empirically that KCS complexity decreases as the models are trained. This is visualized in Fig. 1 for 100 models with up to 100M trainable parameters. Furthermore, we observe that algorithmic complexity is lower in the most significant bit-planes, that it correlates with generalization, and that it can serve as a practical diagnostic for choosing model compression levels, such as precision for post-training quantization.

Our main contributions are:

- We introduce QuBD, which combines quantization with a CTM-based estimator over bit-plane decomposition, making KCS complexity analysis feasible for large, non-binary objects.
- We prove that QuBD yields a smaller estimation gap relative to ideal prefix KCS complexity under bit-plane independence compared to binarization-based methods.
- We use QuBD to study DNN learning dynamics of up to 1B parameters, showing that KCS complexity decreases as models learn and generalize, increases during overfitting, and tracks phenomena like grokking.
- We demonstrate that QuBD complexity serves as a practical diagnostic for model compression, as algorithmic information concentrates in the most significant bit-planes while those with no complexity reduction during training can be safely discarded.

2 Background and Related Work

2.1 Algorithmic Complexity of Neural Networks

Different notions of model compression have been used to explain generalization. For instance, classical learning theory (Vapnik, 1999; Bartlett and Mendelson, 2002) suggests that models with more parameters than data points should overfit, yet such models often generalize well in practice (Zhang et al., 2021; Neyshabur et al., 2017; Belkin et al., 2019). Moreover, a large fraction of model parameters can be removed (Hinton et al., 2014), shared (Ullrich et al., 2017), or quantized (Han et al., 2016) post-training with only small drops in performance.

Since neural networks admit many function-preserving reparametrizations (Dinh et al., 2017), quantities such as curvature or flatness can vary without changing the learned function, limiting how directly they explain generalization behavior (Wilson, 2025; Goldblum et al., 2024). This motivates the KCS complexity view of learning, where complexity and compressibility are studied through the description length of the learned object: the lower the KCS complexity, the shorter the generating description an object admits (Li et al., 2008; Lotfi et al., 2022). From this perspective, Valle-Perez et al. (2019) show that models exhibit a simplicity bias, favoring functions with low KCS complexity, explaining why structured targets are often learned in practice and tend to exhibit redundancy. Moreover, Martin and Hinrichs (2025) find that parameters of a trained model contain measurable correlation structure that evolves during learning and tracks generalization.

Although KCS complexity is non-computable due to the halting problem (Turing, 1936), approximations based on the algorithmic probability of an object (Solomonoff, 1964), computed from large-scale simulations of Turing machines (Soler-Toscano et al., 2014; Zenil et al., 2018), have enabled recent works to assess this perspective for neural networks empirically. Such estimators track structural properties better compared to purely statistical measures like entropy (Zenil et al., 2020), commonly used to explain properties of neural networks (Xu and Raginsky, 2017; Achille and Soatto, 2018). In particular, Sakabe et al. (2026) show that estimated KCS complexity tracks learning dynamics in binarized neural networks more closely than entropy-based measures. That complexity decreases throughout training, capturing changes in the algorithmic structure of the learned function and how such structure shapes the optimization landscape (Bakhtiarifard et al., 2026). However, existing estimators do not scale to modern networks, limiting their applicability and motivating the development of tractable KCS complexity estimators for DNNs.

2.2 Algorithmic Information Theory

The central idea formalized in KCS complexity is that structured objects admit shorter descriptions than truly random objects. For a finite binary object $\mathbf{w} \in \{0,1\}^*$, program p, this is expressed as

$$\mathcal{C}(\mathbf{w}) := \min\{|\mathrm{p}| : U(\mathrm{p}) = \mathbf{w}\}, \tag{1}$$

where U is a fixed universal prefix-free Turing machine and $|\mathrm{p}|$ denotes the length of the program in bits. Since determining whether all shorter programs produce $\mathbf{w}$ and halt is undecidable (Turing, 1936; Davis, 1982), KCS complexity is not computable in general.

Instead, estimates are obtained via algorithmic probability through large-scale simulations of finite Turing machines, using the Coding Theorem (Levin, 1974) as operationalized by Zenil et al. (2015).

The CTM estimates the algorithmic probability of an object by its empirical output frequency across all simulations, and the collection of such estimations is referred to as a CTM table.

BDM (Zenil et al., 2018) extends CTM to larger objects by partitioning objects into smaller blocks, estimating the complexity of each block using pre-computed CTM tables, and accounting for repeated blocks with a logarithmic multiplicity term.

Concretely, let $\mathbf{w}$ be a large object partitioned by a lossless function $\Pi(\mathbf{w}) = (\mathrm{w}_1, \ldots, \mathrm{w}_m)$ and let $\Pi_u(\mathbf{w})$ denote the set of distinct blocks of $\mathbf{w}$. The KCS complexity is then estimated as

$$\mathcal{C}_{\mathrm{BDM}}(\mathbf{w}) = \sum_{\mathrm{w}_j \in \Pi_u(\mathbf{w})} \mathcal{C}_{\mathrm{CTM}}(\mathrm{w}_j) + \log_2 n_{\mathrm{w}_j}, \tag{2}$$

where n_{w_j} is the number of occurrences of $\mathrm{w}_j \in \Pi(\mathbf{w})$ and $\mathcal{C}_{\mathrm{CTM}}(\mathrm{w}_j)$ is the estimated KCS complexity of w_j, derived from its empirical output frequency across Turing machine simulations.

Since CTM tables are defined over binary objects, non-binary objects must first be represented as such prior to partitioning and estimation, which is typically achieved through binarization. Furthermore, the block sizes supported by CTM tables are limited, e.g., (4×4) blocks, which creates a problem when meaningful structures are only revealed at larger scales.

Because scaling CTM tables to larger block sizes is computationally prohibitive and the supported two-dimensional blocks are only defined over binary objects, two fundamental limitations arise. First, the binarized representations may be lossy, discarding useful information about the original object, such that the complexity estimate may correspond to a different target object than intended. Second, the representation and partitioning determine which structures are visible in CTM tables; if meaningful structure is not exposed, it becomes difficult to distinguish genuinely complex objects from objects whose structure is simply obfuscated.

We provide further background on algorithmic complexity in Appx. A. All CTM estimates in this work are from the CTM tables in `pyBDM` (Talaga and Tsampourakis, 2024), with a block size $\pi \in \{(2 \times 2), (3 \times 3), (4 \times 4)\}$. We use the largest $\pi = (4 \times 4)$, unless otherwise stated, as it is the most representative for the objects in consideration.

3 Tractable Algorithmic Complexity Analysis using QuBD

Our proposed KCS complexity estimation method, QuBD, shown in Fig. 2, first quantizes the model weights to a finite alphabet, then represents each quantized weight by its binary expansion across bit-planes. The KCS complexity estimation is then performed by aggregating the CTM-based estimations over all the bit-planes. We show theoretically that under bit-plane independence, this yields a tighter upper-bound estimate of ideal prefix KCS complexity than BDM-based methods that rely on binarization. We verify this empirically and further characterize the conditions under which such estimations remain informative across object scales, given the finite CTM tables.

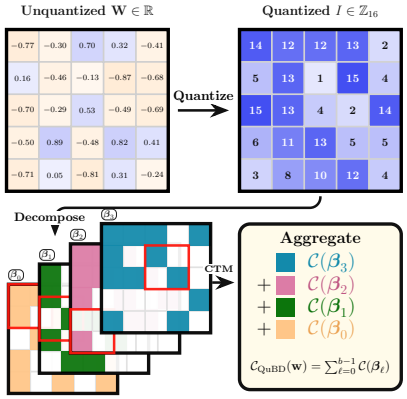


Figure 2: QuBD first quantizes weights into finite codes **(top)**, then decomposes them into aligned bit-planes to aggregate the per-plane KCS estimates **(bottom)**.

Let $\mathbf{w} = (\mathrm{w}_1, \ldots, \mathrm{w}_d) \in \mathbb{R}^d$ denote a d-dimensional real object of model weights. We map $\mathbf{w}$ to a finite alphabet Σ_b using an affine uniform quantizer $Q_b : \mathbb{R}^d \to \Sigma_b^d$, described in Appx. B.1. This yields the discrete object

$$\mathbf{q}_b = Q_b(\mathbf{w}) = \big(q_{b,1}, \ldots, q_{b,d}\big) \in \Sigma_b^d, \text{ with } \Sigma_b := \{0, \ldots, 2^b - 1\}, \tag{3}$$

where each weight q_i, with i being one of d dimensions, is a b-bit symbol of 2^b possible elements. We write $b^\star$ for the target lossless precision, and $\mathbf{w}^\star$ for the corresponding target object. For instance, $b^\star = 32$ for full-precision weights, while the representation is lossless when $b = b^\star$, though precision choices $b \leq 8$ are increasingly relevant for compute-efficient models (Gholami et al., 2022).

Since $\mathbf{q}_b$ must be decomposed into smaller blocks supported by CTM tables, the choice of quantized representation determines which structures are exposed and therefore affects the theoretical gap between the KCS complexity of the target object $\mathbf{w}^\star$ and the estimated object $\mathbf{q}_b$.

Lossless Exposure of Objects. One possible lossless representation of Eq. (3) is the serialized bit string $\mathbf{q}_b^{\text{ser}} \in \{0,1\}^{d\cdot b}$, which for block size π exposes $m = d \cdot b/\pi$ blocks. The KCS complexity of the original non-binary object and the CTM-estimate of the corresponding serialized bit-stream may be different due to the limitations of finite CTM tables (see Sec. 2.2). In contrast, binarization $\mathbf{q}_b^{\text{bin}} \in \{0,1\}^d$ exposes b times fewer blocks but yields an unreliable estimate that differs significantly from that of the target object[2].

The bit-plane representation uses the binary expansion of each quantized weight:

$$q_i = \sum_{\ell=0}^{b-1} 2^\ell \beta_{i,\ell}, \qquad \beta_{i,\ell} \in \{0,1\}, \qquad \boldsymbol{\beta_\ell} := (\beta_{1,\ell}, \ldots, \beta_{d,\ell}) \in \{0,1\}^d \tag{4}$$

where $\beta_{i,b-1}$ denotes the Most Significant Bit (MSB) and $\beta_{i,0}$ denotes the Least Significant Bit (LSB) of q_i. The vector $\boldsymbol{\beta}_\ell$ collects the ℓ-th bit of each of the d symbols, forming the ℓ-th bit-plane. The target object can thus be exposed one bit-plane at a time. The representation remains lossless when $b = b^\star$ since the bit-planes jointly and uniquely determine $\mathbf{q}_b$. The structural information captured by bit-planes has furthermore been shown to be useful for downstream tasks (Nhan et al., 2025). We then define the Quantized Block Decomposition (QuBD) complexity estimate by

$$\mathcal{C}_{\text{QuBD}}(\mathbf{w}) = \sum_{\ell=0}^{b-1} \left(\sum_{\mathrm{w}_j \in \prod(\boldsymbol{\beta}_\ell)} \mathcal{C}_{\text{CTM}}(\mathrm{w}_j) + \log_2 n_{\mathrm{w}_j} \right) = \sum_{\ell=0}^{b-1} \mathcal{C}(\boldsymbol{\beta_\ell}). \tag{5}$$

Interestingly, bit-planes can also be used to reveal structural information progressively, which is a common practice in digital image processing (Chang et al., 1999). Consider discrete objects $q_i \in \mathbf{q}_b$, then for $1 \le k \le b$ define the k-bit MSB prefix index of each q_i as

$$c_i^{(k)} := \left\lfloor \frac{q_i}{2^{b-k}} \right\rfloor = \left\lfloor \frac{c_i^{(k+1)}}{2} \right\rfloor, \quad \text{and} \quad q_i \in \{c2^{b-k}, \ldots, (c+1)2^{b-k} - 1\} \tag{6}$$

where $c_i^{(k)} = c$. For instance, if $b^\star = 4$, $q_i = 13 = 1101_2$, then retaining one bit gives the interval $\{8, \ldots, 15\}$, retaining two bits gives $\{12, \ldots, 15\}$, and retaining three bits gives $\{12, 13\}$.

3.1 Quantized Block Decomposition (QuBD) Complexity

A fixed lossless representation changes the KCS complexity of an object only by an additive constant (Kolmogorov, 1965). In contrast, retaining only the first b bit-planes for $1 \le b < b^\star$ is not lossless. Crucially, even when $b = b^\star$, the object cannot be estimated directly and must be estimated from its decomposition, so its KCS complexity depends on the chosen representation. We consider the change in the target object induced by quantization and ask: *What is the loss in KCS complexity between the target object and the estimated quantized object?*

To formalize this estimation gap, let $\mathbf{w}^\star$ be the target object and consider its decomposition:

$$\mathbf{q}_b := \left\lfloor \frac{\mathbf{w}^\star}{2^{(b^\star - b)}} \right\rfloor = (\mathbf{q}_{b,1}, \ldots, \mathbf{q}_{b,d}), \qquad \mathbf{q}_{b,i} = \left\lfloor \frac{\mathbf{w}_i^\star}{2^{(b^\star - b)}} \right\rfloor = \sum_{\ell=b^\star-b}^{b^\star-1} 2^{\ell-(b^\star-b)} \beta_{i,\ell}^\star. \tag{7}$$

Here $\mathbf{q}_b$ retains exactly the b MSB-planes of $\mathbf{w}^\star$, while discarding the lower $b^\star - b$ bit-planes. For precision $b < b^\star$, define the omitted lower-bit residuals $\delta_b = (\delta_{b,1}, \ldots, \delta_{b,d})$ by

$$\delta_b := \mathbf{w}^\star - 2^{(b^\star - b)} \mathbf{q}_b. \tag{8}$$

Here, $\delta_{b,i} = \mathbf{w}_i^\star - 2^{(b^\star-b)}\mathbf{q}_{b,i}$ and $0 \le \delta_{b,i} < 2^{b^\star - b}$ for each i. Since $\mathbf{q}_b$ and δ_b capture the retained and omitted b bit-planes, respectively, the target object can be estimated as the composite $(\mathbf{q}_b, \delta_b)$.

[2] When binarization is applied through the sign function, it defines a different target object. The additive-constant invariance of KCS complexity applies only to lossless changes of description of the same object (Kolmogorov, 1965); a sign map is many-to-one and therefore falls outside that setting. However, we include it in this work for completeness.

We analyze the theoretical gap between $\mathbf{w}^\star$ and $(\mathbf{q}_b, \delta_b)$ using standard information-symmetry identities of KCS complexity from Li et al. (2008), summarized in Appx. C.1. An example of bit-plane decomposition is provided in Appx. B.2.

Proposition 3.1 (Residual Decomposition). *For a d-dimensional target object $\mathbf{w}^\star \in \mathbb{R}^d$ with precision $1 \leq b \leq b^\star$, the ideal prefix KCS complexity is*

$$\mathcal{C}(\mathbf{w}^\star) = \mathcal{C}(\mathbf{q}_b) + R_b + O(\log(b^\star d)), \qquad R_b := \mathcal{C}(\delta_b \mid \mathbf{q}_b), \tag{9}$$

where $0 \leq R_b \leq (b^\star - b)d + O(1)$. Estimating $\mathbf{w}^\star$ from $\mathbf{q}_b$ changes the complexity due to the residual information R_b up to a logarithmic term. If $b = b^\star$, then $\delta_b = 0$, $R_b = O(1)$, and the decomposition offers a target object differing only by a constant complexity. Proof in Appx. C.2.

The inequalities in Prop. 3.1 are necessarily tight since $R_b \to (b^\star - b)d + O(1)$ when there is information about the discarded bit-planes in the retained ones. On the other hand, $R_b = O(1)$ when there is no such information. So estimating $(\mathbf{q}_b, \delta_b)$ instead results in only a logarithmic error in bit length and in the residual information from the omitted bits. The following analysis demonstrates that the residual diminishes as additional bit-planes are incorporated, improving the upper-bound on the estimation gap when assuming independence across bit-planes.

Proposition 3.2 (Bit-Plane Residual Loss). *For $1 \leq b < b^\star$ and independent bit-planes $\boldsymbol{\beta}_\ell$, let*

$$\boldsymbol{P}_{b+1} := (\beta^\star_{1,(b^\star-b)-1}, \ldots, \beta^\star_{d,(b^\star-b)-1}) \in \{0,1\}^d, \tag{10}$$

where $\boldsymbol{P}_{b+1}$ is the next MSB plane after the b retained in $\mathbf{q}_b$. Then $\mathbf{q}_{b+1} = 2\mathbf{q}_b + \boldsymbol{P}_{b+1}$, and

$$R_b \geq R_{b+1} - O(\log(b^\star d)). \tag{11}$$

Thus, adding the $(b+1)$-th bit-plane cannot increase the residual loss, and more importantly, the residual loss decreases whenever bit-planes are independent and not identical. Proof in Appx. C.3.

Repeated application of Prop. 3.2 shows that if $b = b^\star$, only a constant error remains; we refer to Cor. C.1 and its proof in Appx. C.3.1. Importantly, when $b > 1$, the bit-plane decomposition yields a smaller residual than one-bit binarization under the independence assumption. And the impact of the quantization level on the decomposition does not scale beyond $O(\log(b^\star d))$. For clarity, note that the decomposition overhead remains because the refinement does not guarantee the same ordering.

We validate our findings in Appx. C.4 (Fig. 9) by comparing the estimation gap to the reference CTM-table complexity across different representations. This shows that this gap narrows as QuBD retains more bit-planes. For clarity, validating directly on other objects, for instance, model weights, would confound estimator saturation with the structure induced by training noted by Valle-Perez et al. (2019). We next analyze how QuBD complexity scales with CTM-tables, providing more informative estimates at larger scales.

3.2 Saturation in Finite-state Machines

Let the binary block size π be fixed, and define $S_\pi \subseteq \{0,1\}^\pi$ as the finite CTM support. For an exposure $z_{b,r} = \phi(\mathbf{q}_b)$, the number of blocks m is determined after the quantized object is exposed by representation r and partitioned into binary blocks, $\Pi_\pi(z_{b,r}) = (\mathrm{w}_1, \ldots, \mathrm{w}_m)$, each of size π. The same d quantized symbols may yield a different number of binary blocks depending on r. Define $S^{(b)}_{\pi,r} \subseteq S_\pi$ as the set of supported blocks reachable under this exposure, and let $a_\pi(z_{b,r})$ denote the number of reachable blocks that appear among $(\mathrm{w}_1, \ldots, \mathrm{w}_m)$. We then define

$$H^{(b)}_{\pi,r} := |S^{(b)}_{\pi,r}|, \qquad \sigma_{\pi,r}(m) := \frac{\mathbb{E}[a_\pi(z_{b,r})]}{H^{(b)}_{\pi,r}}. \tag{12}$$

Here, $\sigma_{\pi,r}(m)$ represents the expected fraction of reachable support that appears after m exposed blocks. The estimate is considered $(1-\epsilon)$-saturated for some $\epsilon \in [0,1]$ when $\sigma_{\pi,r}(m) \geq 1 - \epsilon$. Additional details are provided in Appx. D.1.

Proposition 3.3 (Finite-table Saturation). *Assume the m exposed blocks are sampled independently and uniformly from $S^{(b)}_{\pi,r}$. Then*

$$\sigma_{\pi,r}(m) = 1 - \left(1 - \frac{1}{H^{(b)}_{\pi,r}}\right)^m \approx 1 - e^{-m/H^{(b)}_{\pi,r}}, \qquad m_\epsilon \gtrsim H^{(b)}_{\pi,r} \log \frac{1}{\epsilon}, \tag{13}$$

where m_ϵ is the number of exposed blocks expected for $(1-\epsilon)$-saturation. Proof in Appx. D.2.

By Prop. 3.3, exposure r saturates when $m_r \approx H^{(b)}_{\pi,r} \log(1/\epsilon)$. For d quantized symbols, $m_{\text{bin}} = d/\pi$, $m_{\text{ser}} = d \cdot b/\pi$, and $m_{\text{plane}} = d/\pi$ per bit-plane, which yields

$$d_{\text{bin}} \gtrsim \pi H_{\pi,\text{bin}} \log \frac{1}{\epsilon}, \qquad d_{\text{ser}} \gtrsim \frac{\pi H^{(b)}_{\pi,\text{ser}}}{b} \log \frac{1}{\epsilon}, \qquad d_{\text{plane}} \gtrsim \pi H_{\pi,\text{plane}} \log \frac{1}{\epsilon}. \tag{14}$$

While binarization saturates more slowly, it incurs a larger residual loss. In contrast, serialization is lossless but saturates earlier, as it exposes b times more blocks to a single finite CTM table. Bit-plane exposure estimates the same target object by scaling the support according to the number of bit-planes while evaluating local structures independently across its bit-planes. The corresponding saturation rates are illustrated in Appx. D.3 (Fig. 10).

When Representations Fail. The problem with the saturation effect (Sec. 3.2) is that $a_\pi(z_{b,r})$ stops growing while m keeps increasing, to the point where only multiplicity is counted (recall Eq. (2)). While this is reasonable for simple objects, it is not for structurally rich or large objects, where additional local structure can no longer be captured. Hence, the estimate becomes uninformative, similar to the entropy collapse described in Sakabe et al. (2026).

Representation failure is illustrated in Fig. 3, which compares a structured 8-ary object $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_d)$ with its partially permuted counterpart $\mathbf{x}_\rho$ using $\pi = (4 \times 4)$, where ρ is the fraction of positions permuted. Let $n_a := |\{i : \mathrm{x}_i = a\}|$ be the number of occurrences of symbol $a \in \Sigma_b$ in $\mathbf{x}$. Since the permutation preserves all symbol counts and alters only their positions, any difference in complexity estimates must arise from the change in arrangement. The information required to identify one such arrangement is $\log_2(d!/\prod_a n_a!)$ bits (Cover, 1999; Li et al., 2008), which serves as a reference for the complexity added by the permutation. The values in Fig. 3 are normalized by the object type class described in Cover (1999). An informative estimator should assign higher complexity to $\mathbf{x}_\rho$ than to $\mathbf{x}$, with the gap increasing in ρ. If the KCS complexity gap $\mathcal{C}(\mathbf{x}_\rho) - \mathcal{C}(\mathbf{x})$ collapses toward zero, the estimator fails to identify structure. As shown in Fig. 3, QuBD maintains a larger and more stable estimation gap than serialization, one-bit, and sign-binarization. Similar results are found for CTM tables of $\pi = (3 \times 3)$ and $\pi = (3 \times 3)$, illustrated in Appx. D.1 (Fig. 11) Importantly, in the saturated regime, retaining even a small fraction of the reference gap still yields a distinguishable measure of lower- and higher-complexity models. This regime is shown in Appx. D.3 (Fig. 12), and additional results in Appx. E.4, finding similar improvements in KCS estimation with QuBD.

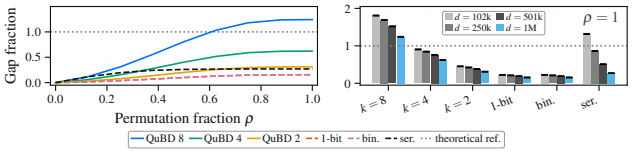


Figure 3: Complexity gap between estimations of an ordered 8-ary object and its permutation, where ρ is the fraction of symbols d permuted. The gap is shown **(left)** for an object with $d = 1M$ symbols as ρ increases, and **(right)** across object sizes for different methods when $\rho = 1$.

4 Characterizing Learning in Deep Neural Networks

We normalize the QuBD complexity estimate of an object $\mathbf{w}$ relative to a random object of the same length $\tilde{\mathbf{w}}$, denoted $\Delta\mathcal{C}_{\text{QuBD}}$, for easier interpretation:

$$\Delta\mathcal{C}_{\text{QuBD}} = \frac{\mathcal{C}_{\text{QuBD}}(\mathbf{w})}{\mathcal{C}_{\text{QuBD}}(\tilde{\mathbf{w}})}. \tag{15}$$

A truly random object has $\Delta\mathcal{C}_{\text{QuBD}} \approx 1$, whereas an algorithmically simple object[3] has $\Delta\mathcal{C}_{\text{QuBD}} \approx 0$.

Learning Reduces Algorithmic Complexity of Networks. Neural network weights are initialized randomly and, from an algorithmic complexity perspective, constitute highly complex objects because they lack structure and therefore cannot be compressed. This raises a natural question: *What happens to the algorithmic complexity of the weights as networks train?*

[3]Strictly speaking, $\Delta\mathcal{C}_{\text{QuBD}} \approx 0$ holds only for sufficiently large objects, as even maximally simple objects such as an all-zero string retain some complexity. For objects of sizes (8×8), (16×16), (32×32), $\Delta\mathcal{C}_{\text{QuBD}}$ for an all-zero string is 0.19, 0.05, and 0.01, respectively.

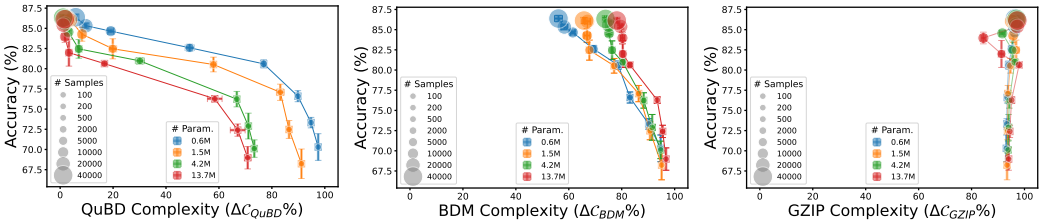


Figure 4: Complexity of trained models compared to random initialization when using QuBD, BDM, and GZIP-based estimations, measured as $\Delta\mathcal{C}$ based on Eq. 15. Four MLPs with different numbers of trainable parameters ($0.6M - 13.7M$) are trained with increasing number of training data points and validated on the same test set. QuBD complexity tracks the connection between learning and algorithmic complexity better than BDM or GZIP.

In Fig. 4 **(left)**, we present clear empirical evidence that the algorithmic complexity of neural networks measured using QuBD (Sec. 3) decreases with training. Multi-layered perceptrons (MLPs) with varying numbers of trainable parameters, ranging from $0.6M$ to $13.7M$, are trained on the Fashion-MNIST dataset (Xiao et al., 2017) with increasing number of training samples and evaluated on the same validation set. For each model, the test accuracy improves with more training data, and the normalized QuBD complexity, $\Delta\mathcal{C}_{\text{QuBD}}$, decreases accordingly, indicating that the improvements in test performance correlate with reductions in algorithmic complexity. This effect is most pronounced for the largest model, whose QuBD complexity reduces to approximately 5% of its value at initialization. To investigate whether there is an effect of the number of symbols d exposed for varying object sizes, for instance, in Fig. 1, we compute the Spearman correlation of the relationship between model size and $\Delta\mathcal{C}_{\text{QuBD}}$ and find a weak relationship ($|\rho| < 0.2$) for all bit-planes $\ell = 0, \ldots, 7$, illustrated in Appx F.3 (Fig. 17). This suggests that the ordering is stable across model sizes, consistent with the saturation effect characterized for increasing object sizes in Sec. 3.2.

Fig. 4 **(center)** and Fig. 4 **(right)** report two commonly used complexity measures for comparison: BDM with weight binarization (Sakabe et al., 2026; Bakhtiarifard et al., 2026), and raw weight compression using GZIP. As clearly seen in these plots, and further validated in Appx. F.4 (Fig. 18), neither measure correlates meaningfully with learning, in contrast to QuBD. The results shown here are when using 8 bit-planes for QuBD. Additional experimental details are provided in Appx. E.2.

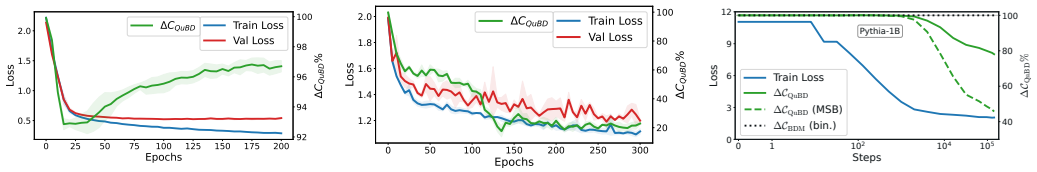


Figure 5: Learning curves along with $\Delta\mathcal{C}_{\text{QuBD}}$ evolution for MLP **(left)**, Tiny-ViT **(center)** and Pythia-1B **(right)**. Loss and $\Delta\mathcal{C}_{\text{QuBD}}$ for MLP and Tiny-ViT are reported over three random repeats.

Algorithmic Complexity Tracks Generalization. Fig. 5 **(right)** probes into the evolution of algorithmic complexity reduction during training for an MLP trained on Fashion-MNIST (Xiao et al., 2017) and Tiny-ViT (Wu et al., 2022) trained on CIFAR-10 (Krizhevsky, 2009). In both cases, $\Delta\mathcal{C}_{\text{QuBD}}$ tracks the generalization performance. The most interesting observation is in Fig. 5 **(left)**, where the algorithmic complexity increases as the network starts to overfit. As overfitting is generally attributed to memorization, this suggests that the network is no longer simplifying its underlying algorithmic structure, leading to an increase in algorithmic complexity. For the Tiny-ViT case, QuBD complexity follows the learning curves, indicating that learning continues to reduce the algorithmic complexity of networks. Additional experimental details are provided in Appx. E.3. We further study whether the relation between training dynamics and QuBD complexity extends to frontier AI models, using the Pythia-1B large language model (LLM), which was trained using The Pile dataset comprising about 300B tokens (Biderman et al., 2023). Similar to Fig. 5 **(left)**, we observe that the QuBD complexity decreases, and the network continues simplification after the loss plateaus, illustrated in Appx. E.4 (Fig. 13).

Algorithmic Complexity during Grokking. Grokking (Power et al., 2021) is a phenomenon observed in over-parameterized neural networks, where the networks generalize well past the point of overfitting, defying conventional machine learning principles (Belkin et al., 2019). This raises the question: *What happens to the algorithmic complexity of neural networks in such regimes?*

To study this, we set up grokking experiments for the modulo operator as described in Power et al. (2021), and measure the QuBD complexity during training, as shown in Fig. 6. The network reaches zero training error within the first few epochs, whereas the validation loss drops to zero after about 1000 epochs, consistent with the findings reported in Power et al. (2021) and other works. The algorithmic complexity, measured as the reduction in QuBD complexity, remains high during the high-validation-loss regime and decreases when the validation loss has reached zero. It continues to decrease even after the validation loss is zero, indicating further algorithmic simplification of the neural network. This is in contrast to the overfitting example demonstrated in Fig. 5 **(left)**.

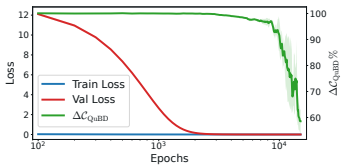


Figure 6: Grokking for the Modulo operator for $P = 97$.

Algorithmic Information Resides in Higher Bit-Planes. When bit-planes are independent, QuBD complexity offers tighter estimates on ideal prefix KCS complexity of non-binary objects compared to BDM, due to Prop. 3.1 and Prop. 3.2. Adding bit-planes captures additional algorithmic information, but saturates beyond a few. This is illustrated in Fig. 1, where we show the relative complexity reduction achieved by the QuBD method per bit-plane. For the higher bit-planes, in particular the MSB, $\Delta\mathcal{C}_{\text{QuBD}}$ is lower for all 100 models with $1M$ to $100M$ trainable parameters, whereas the lower bit-planes show no reduction in complexity. Additional experimental details can be found in Appx. E.5.

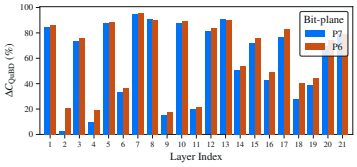


Figure 7: QuBD complexity ratio $\Delta\mathcal{C}_{\text{QuBD}}$ per layer for ResNet-18 for the two MSB-planes (P7, P6).

To use QuBD for models with high-dimensional weight tensors, e.g., 4D convolutional layers, the weight tensors must first be preprocessed into 2D matrices prior to estimation, as described in Appx. B.3. Using this, we examine $\Delta\mathcal{C}_{\text{QuBD}}$ per layer of ResNet-18 (Fig. 7) and ResNet-50 (Appx. F.5) for the two MSB-planes (P7, P6). It is well established that different layers in a neural network have different importance, and that a uniform level of pruning or quantization can hamper performance. This variation across layers is clearly reflected in the QuBD-complexity reduction for both ResNet models (Wenshøj et al., 2025). While there is no clear trend, some inner layers appear to exhibit significant complexity reduction. This could also explain why these inner layers are better suited for compression than input or output layers.

We have seen that lower bit-planes have less algorithmic information, leading to the question: *If the lower bit-planes contain less algorithmic information, can these planes be discarded?*

QuBD Complexity as a Diagnostic for Compression. To test this hypothesis, we measure the QuBD complexity of five full precision (FP32) models: ResNet-18, ResNet-50, ViT, EfficientNet, and MobileNet, trained on ImageNet (Deng et al., 2009) using 16 bit-planes. We then perform post-training quantization (PTQ) on the same models with weight precision ranging from 1-bit to 8-bit and evaluate the quantized models on the ImageNet validation set. The first observation in Fig. 8 **(left)** is that $\Delta\mathcal{C}_{\text{QuBD}} \approx 100\%$ for the lower bit-planes, indicating they are close to random and might not contain any useful algorithmic information. Notably, meaningful complexity reduction is concentrated in the top five bit-planes. This correlates with the PTQ performance in Fig. 8 **(right)**, which does not show large performance drops after 4-to-6-bit quantization, suggesting that the remaining bit-planes can be

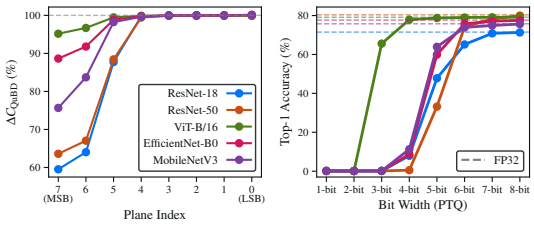


Figure 8: QuBD complexity $\Delta\mathcal{C}_{\text{QuBD}}$ per bit-plane **(left)** and top-1 PTQ accuracy on ImageNet **(right)** for five pretrained models under 1-to-8-bit quantization. Dashed lines indicate FP32 baseline accuracy.

discarded without a performance penalty. Measuring KCS complexity using the QuBD method can serve as a diagnostic to determine PTQ quantization levels (Fig. 8) or which layers to prune (Fig. 7). Additional experimental details are provided in Appx. E.6.

Influence of Quantization on QuBD Complexity. Estimating the QuBD complexity entails a quantization step as described in Sec. 3. One concern about this transformation is whether the resulting KCS complexity is related to the actual model weights or whether quantization has an adverse impact. We verify this by comparing standard FP32 training with quantization-aware training (QAT) at 32-bit precision, and measuring the QuBD complexity using 8 bit-planes. Tab. 1 shows that there is no significant difference between the original model and the QAT model, both in terms of accuracy and QuBD complexity reduction. These are key differences compared to other methods that rely on binarizing the weights of an FP32 network before estimating KCS complexity, as the two objects (FP32 and binary versions) may not be comparable in those settings. This is further exemplified in Appx. E.4 where the QuBD MSB-plane does not coincide with the sign-pattern found when binarizing.

Table 1: Comparison of FP32 and QAT with 32 bits.

Precision	**Accuracy**	$\Delta\mathcal{C}_{\text{QuBD}}$
FP32	86.13 ± 0.46	95.51 ± 0.90
QAT-32	86.22 ± 0.44	96.39 ± 0.29

We further verify estimation robustness to the effect of reshaping in Appx. B.3, and the weight distribution incurred by percentile-clipping in Appx. F.2 (Tab. 3). This can change the structures exposed to the CTM table. We find, however, negligible impact for both effects, with the trends observed in Fig. 4 persisting across all model scales and sample budgets. Importantly, no other assumptions on architecture, layer shapes, or the distribution, for instance, its symmetry, of the trained weights or their extrema are imposed.

Limitations. We aggregate the algorithmic complexity per bit-plane in QuBD, which assumes independence across bit-planes. This might not be true as there could be mutual algorithmic information shared across bit-planes, which implies QuBD can be overestimating the underlying algorithmic complexity. While QuBD provides a tighter upper-bound estimate on the ideal prefix complexity than BDM, accounting for the mutual information between bit-planes could yield even tighter KCS complexity estimations. Further, we demonstrated that QuBD can identify redundant bit-planes and that weight structures can be pruned based on QuBD complexity, suggesting its potential as a practical diagnostic for both PTQ level selection and pruning. A natural extension is to validate these applications within thorough compression experiments and benchmark them against established methods. For instance, measuring saturation in individual bit-plane exposures in practice. As we find consistent evidence for complexity reduction estimated by QuBD, the checkpoints used for the larger Pythia models consider just one epoch, so they do not extend to the overfitting regime and over random initializations, but do coincide with the grokking experiments.

5 Conclusions

Several attempts have been made to explain learning in deep neural networks. We contribute a novel view based on algorithmic information theory, building on some of the recent works (Sakabe et al., 2026; Bakhtiarifard et al., 2026). We operationalize KCS complexity estimation using the QuBD method. To our knowledge, this is the first scalable method to reliably estimate KCS complexity of large, non-binary objects. We have shown theoretically that the QuBD method under bit-plane independence yields a tighter upper-bound estimate on ideal prefix KCS complexity compared to binarization. Using this algorithmic complexity analysis tool, we have shown that learning in deep neural networks induces structure, or equivalently, reduces algorithmic complexity. We have shown, across a wide variety of experiments, that QuBD complexity correlates with generalization and phenomena such as grokking. We have demonstrated that higher bit-planes carry more algorithmic structure than lower bit-planes. This can be used to identify redundant model capacity, for example, by choosing appropriate PTQ levels. Algorithmic information theory is a powerful paradigm that has been creatively applied to prove Darwinian evolution (Chaitin, 2013), and we believe it can also be useful for explaining deep learning.

Acknowledgments and Disclosure of Funding

All authors acknowledge funding from the European Union's Horizon Europe Research and Innovation Action program under grant agreements No. 101070284, No. 101070408 and No. 101189771. PB acknowledges funding from Stibofonden, Knud Højgaards Fond, William Demant Fonden, and Familien Hede Nielsens Fond. SW and RS acknowledge funding from the Independent Research Fund Denmark (DFF) under grant agreement No. 4307-00143B. The authors thank members of SAINTS Lab for valuable discussions.

References

A. Achille and S. Soatto. Emergence of Invariance and Disentanglement in Deep Representations. *Journal of Machine Learning Research*, 2018.

A. Alzahrani. Utility of Kolmogorov Complexity Measures: Analysis of L2 Groups and L1 Backgrounds. *PLOS ONE*, 2024.

L. F. W. Anthony, B. Kanding, and R. Selvan. Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models. In *International Conference on Machine Learning (ICML) Workshop on Challenges in Deploying and Monitoring Machine Learning Systems*, 2020.

P. Bakhtiarifard, T. Chen, J. Wenshøj, E. B. Dam, and R. Selvan. Algorithmic Simplification of Neural Networks with Mosaic-of-Motifs. *arXiv preprint arXiv:2602.14896*, 2026.

P. L. Bartlett and S. Mendelson. Rademacher and Gaussian Complexities: Risk Bounds and Structural Results. *Journal of Machine Learning Research*, 2002.

B. R. Bartoldson, B. Kailkhura, and D. Blalock. Compute-Efficient Deep Learning: Algorithmic Trends and Opportunities. *Journal of Machine Learning Research*, 2023.

M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling Modern Machine Learning Practice and the Classical Bias–Variance Trade-Off. *National Academy of Sciences*, 2019.

S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, et al. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. In *International Conference on Machine Learning (ICML)*, 2023.

G. J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the Association for Computing Machinery*, 1966.

G. J. Chaitin. Algorithmic Information Theory. *IBM Journal of Research and Development*, 1977.

G. J. Chaitin. *Proving Darwin: Making Biology Mathematical*. Vintage, 2013.

C.-C. Chang, F.-C. Shiue, and T.-S. Chen. A New Scheme of Progressive Image Transmission Based on Bit-Plane Method. In *Fifth Asia-Pacific Conference on... and Fourth Optoelectronics and Communications Conference on Communications*. IEEE, 1999.

T. M. Cover. *Elements of Information Theory*. Wiley–Interscience, 1999.

M. Davis. *Computability & Unsolvability*. Courier Corporation, 1982.

J.-P. Delahaye and H. Zenil. Numerical Evaluation of Algorithmic Complexity for Short Strings: A Glance Into the Innermost Structure of Randomness. *Applied Mathematics and Computation*, 2012.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp Minima Can Generalize For Deep Nets. In *International Conference on Machine Learning (ICML)*, 2017.

J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2019.

J.-L. Gailly and M. Adler. Gnu gzip. *GNU Operating System*, 1992. URL https://www.gnu.org/.

A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A Survey of Quantization Methods for Efficient Neural Network Inference. In *Low-Power Computer Vision*. Chapman & Hall/CRC, 2022.

M. Goldblum, M. A. Finzi, K. Rowan, and A. G. Wilson. Position: The No Free Lunch Theorem, Kolmogorov Complexity, and the Role of Inductive Biases in Machine Learning. In *International Conference on Machine Learning (ICML)*, 2024.

Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv preprint arXiv:1412.6115*, 2014.

S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)*, 2016.

G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. In *Deep Learning and Representation Learning Workshop in Conjunction with NeurIPS*, 2014.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

A. N. Kolmogorov. Three Approaches to the Quantitative Definition of Information. *Problems of Information Transmission*, 1965.

L. G. Kraft. *A Device for Quantizing, Grouping, and Coding Amplitude-Modulated Pulses*. PhD thesis, Massachusetts Institute of Technology, 1949.

A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009.

L. A. Levin. Laws of Information Conservation (Nongrowth) and Aspects of the Foundation of Probability Theory. *Problemy Peredachi Informatsii*, 1974.

M. Li, P. Vitányi, et al. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 2008.

T. Liu and L. Simine. DeltaGzip: Computing Biopolymer–Ligand Binding Affinity via Kolmogorov Complexity and Lossless Compression. *Journal of Chemical Information and Modeling*, 2024.

S. Lotfi, M. Finzi, S. Kapoor, A. Potapczynski, M. Goldblum, and A. G. Wilson. PAC-Bayes Compression Bounds so Tight that they can Explain Generalization. *Neural Information Processing Systems (NeurIPS)*, 2022.

J. A. T. Machado, J. M. Rocha-Neves, F. Azevedo, and J. P. Andrade. Advances in the Computational Analysis of SARS-COV2 Genome. *Nonlinear Dynamics*, 2021.

C. H. Martin and C. Hinrichs. SETOL: A Semi-Empirical Theory of (Deep) Learning. *arXiv preprint arXiv:2507.17912*, 2025.

B. McMillan. Two Inequalities Implied by Unique Decipherability. *IRE Transactions on Information Theory*, 2003.

M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort. A White Paper on Neural Network Quantization. *arXiv preprint arXiv:2106.08295*, 2021.

B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring Generalization in Deep Learning. In *Neural Information Processing Systems (NeurIPS)*, 2017.

L. T. Nhan, L. T. Duong, P. N. Nam, and T. C. Thang. Improvement of Spiking Neural Network with Bit Planes and Color Models. *IEEE Access*, 2025.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An Imperative Style, High-Performance Deep Learning Library. *Neural Information Processing Systems (NeurIPS)*, 2019.

D. Peptenatu, I. Andronache, H. Ahammer, R. Taylor, I. Liritzis, M. Radulovic, B. Ciobanu, M. Burcea, M. Perc, T. D. Pham, et al. Kolmogorov Compression Complexity May Differentiate Different Schools of Orthodox Iconography. *Scientific Reports*, 2022.

A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets. In *International Conference on Learning Representations (ICLR) Workshop on Mathematical Reasoning in General Artificial Intelligence*, 2021.

J. Rigau, M. Feixas, and M. Sbert. Conceptualizing Birkhoff's Aesthetic Measure Using Shannon Entropy and Kolmogorov Complexity. In *Third Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, 2007.

E. Y. Sakabe, F. S. Abrahão, A. Simões, E. Colombini, P. Costa, R. Gudwin, and H. Zenil. Binarized Neural Networks Converge Toward Algorithmic Simplicity: Empirical Support for the Learning-as-Compression Hypothesis. *Frontiers in Computational Neuroscience*, 2026.

J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos. Compute Trends Across Three Eras of Machine Learning. In *International Joint Conference on Neural Networks*. IEEE, 2022.

J. Seward. bzip2 and libbzip2: A Program and Library for Data Compression, 1996. URL http://www.bzip.org.

R. Shwartz-Ziv and N. Tishby. Opening the Black Box of Deep Neural Networks via Information. *arXiv preprint arXiv:1703.00810*, 2017.

F. Soler-Toscano, H. Zenil, J.-P. Delahaye, and N. Gauvrit. Calculating Kolmogorov Complexity from the Output Frequency Distributions of small Turing Machines. *Public Library of Science ONE*, 2014.

R. J. Solomonoff. A Preliminary Report on a General Theory of Inductive Inference. Technical report, Zator Company Cambridge, Massachusetts, 1960.

R. J. Solomonoff. A Formal Theory of Inductive Inference. Part I. *Information and Control*, 1964.

E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

S. Talaga and K. Tsampourakis. PyBDM: Python interface to the Block Decomposition Method (0.1.0), 2024.

N. Tishby, F. C. Pereira, and W. Bialek. The Information Bottleneck Method. *In Proceedings at Allerton Conference on Communication*, 2000.

A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *In Proceedings of the London Mathematical Society*, 1936.

K. Ullrich, E. Meeds, and M. Welling. Soft Weight-Sharing for Neural Network Compression. In *International Conference on Learning Representations (ICLR)*, 2017.

G. Valle-Perez, C. Q. Camargo, and A. A. Louis. Deep Learning Generalizes because the Parameter-Function Map is Biased Towards Simple Functions. In *International Conference on Learning Representations (ICLR)*, 2019.

V. N. Vapnik. An Overview of Statistical Learning Theory. *IEEE Transactions on Neural Networks*, 1999.

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.

R. Wightman. PyTorch Image Models. https://github.com/rwightman/pytorch-image-models, 2019.

A. G. Wilson. Position: Deep Learning is Not So Mysterious or Different. In *International Conference on Machine Learning (ICML) Position Paper Track*, 2025.

K. Wu, J. Zhang, H. Peng, M. Liu, B. Xiao, J. Fu, and L. Yuan. TinyViT: Fast Pretraining Distillation for Small Vision Transformers. In *European Conference on Computer Vision (ECCV)*, 2022.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

A. Xu and M. Raginsky. Information-Theoretic Analysis of Generalization Capability of Learning Algorithms. *Neural Information Processing Systems (NeurIPS)*, 2017.

H. Zenil, F. Soler-Toscano, J.-P. Delahaye, and N. Gauvrit. Two-Dimensional Kolmogorov Complexity and Validation of the Coding Theorem Method by Compressibility. *PeerJ Computer Science*, 2015.

H. Zenil, S. Hernández-Orozco, N. A. Kiani, F. Soler-Toscano, A. Rueda-Toicen, and J. Tegnér. A Decomposition Method for Global Evaluation of Shannon Entropy and Local Estimations of Algorithmic Complexity. *Entropy*, 2018.

H. Zenil, N. Kiani, F. Abrahão, and J. Tegnér. *Algorithmic Information Dynamics*. Scholarpedia, 2020.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding Deep Learning (Still) Requires Rethinking Generalization. *Communications of the Association for Computing Machinery*, 2021.

A Additional Details on Algorithmic Information Theory

For completeness, the definitions introduced in the main text are restated here, with additional detail provided where necessary. Kolmogorov-Chaitin-Solomonoff (KCS) complexity (Kolmogorov, 1965; Chaitin, 1966; Solomonoff, 1960) formalizes the principle that structured objects can be described more concisely than random objects. For a finite object $\mathbf{w} \in \{0,1\}^*$ and a program p, this relationship is expressed as:

$$\mathcal{C}(\mathbf{w}) := \min\{|\mathrm{p}| : U(\mathrm{p}) = \mathbf{w}\}. \tag{16}$$

Here, p is executed on a fixed universal prefix Turing machine U. In general, Eq. (16) is not computable because determining whether all shorter programs produce $\mathbf{w}$ and halt, fail, or run indefinitely is undecidable. This is known as the *halting problem* (Turing, 1936; Davis, 1982).

A related measure to the KCS complexity of an object is its algorithmic probability (Solomonoff, 1964), which assigns a probability to each possible output produced by a program:

$$\mathbb{P}_U(\mathbf{w}) := \sum_{\mathrm{p}:U(\mathrm{p})=\mathbf{w}} 2^{-|\mathrm{p}|} \leq 1. \tag{17}$$

Here $\mathbb{P}_U(\mathbf{w})$ is the probability that a random program produces $\mathbf{w}$ and halts. The last inequality follows from the Kraft–McMillan inequality (Kraft, 1949; McMillan, 2003), since the halting programs form a prefix code, i.e., no program is the start of another program.

To help gain intuition into approaches for estimating KCS using algorithmic probability, consider an object $\mathbf{w}$ produced by a program p, and suppose there are m distinct programs of length $|\mathrm{p}|$ such that $U(\mathrm{p}) = \mathbf{w}$. Then $\mathbb{P}_U(\mathbf{w}) \geq m \cdot 2^{-|\mathrm{p}|}$ by Eq. 17 and $\mathcal{C}(\mathbf{w}) \leq |\mathrm{p}|$, expressing the upper bound on its KCS complexity. For instance, if $|\mathrm{p}| = 4$, each such program contributes $2^{-4} = 1/16$, so $\mathbb{P}_U(\mathbf{w}) \geq m/16$. Thus, the existence of one short program gives a lower bound on the algorithmic probability of $\mathbf{w}$. More generally, a larger output probability suggests the existence of shorter (program) descriptions.

An important caveat is that prefix programs are self-delimiting, meaning their description also contains information about where they end. Therefore, estimating complexity from a decomposed view of an object is not the same as estimating the entire object directly. In particular, suppose the partition $\Pi(\mathbf{w}) = (\mathrm{w}_1, \ldots, \mathrm{w}_m)$ divides the object $\mathbf{w}$ into m ordered blocks w_j. Then,

$$\mathcal{C}(\mathrm{w}_1, \ldots, \mathrm{w}_m) = \sum_{j=1}^{m} \mathcal{C}(\mathrm{w}_j) - I_\Pi(\mathrm{w}_1, \ldots, \mathrm{w}_m) + O(\log_2 m), \tag{18}$$

where $\mathcal{C}(\mathrm{w}_j)$ denotes the prefix KCS complexity of w_j and I_Π denotes the algorithmic information shared across blocks in the partition Π. An estimate that sums local complexities, therefore, ignores part of this shared structure, and the $O(\log_2 m)$ term accounts for the overhead needed to delimit and reconstruct the ordered decomposition of $\mathbf{w}$. This induces a representation and partition error, which is why both matter. Some useful identities about these errors are summarized in Appx. C.1.

The Coding Theorem (Levin, 1974) connects the output probability of an object to its complexity by

$$\mathcal{C}(\mathbf{w}) = -\log_2 \mathbb{P}_U(\mathbf{w}) + O(1), \tag{19}$$

where $O(1)$ is a constant factor depending only on U, e.g., changing to a machine that uses different instructions adds a constant cost irrespective of $\mathbf{w}$. The central idea behind the Coding Theorem, that objects that appear often as outputs are of low complexity, motivated the Coding Theorem Method (CTM) for numerically estimating KCS complexity via large-scale simulations of small Turing machines (Delahaye and Zenil, 2012; Soler-Toscano et al., 2014; Zenil et al., 2015).

Simulating Turing Machines. Fix integers $n, k, T \in \mathbb{N}$ and let $\mathcal{M}_{n,k}^T$ denote the set of all n-state, k-symbol Turing machines under a fixed encoding, each run for at most T steps. Since n, k, T are fixed, $\mathcal{M}_{n,k}^T$ is finite, and all machines in the class can, in principle, be simulated. Let $\rho_T(A) = \mathbf{w} \in \mathcal{W}_T$ denote the output of A whenever A halts within T steps. CTM estimates the algorithmic probability of $\mathbf{w}$ by its empirical frequency

$$\mathrm{Dist}_{n,k}^T(\mathbf{w}) = \frac{|\{A \in \mathcal{M}n, k^T : \rho_T(A) = \mathbf{w}\}|}{|\{A \in \mathcal{M}_{n,k}^T : A \text{ halts within } T \text{ steps}\}|}, \tag{20}$$

and the corresponding complexity estimate is $\mathcal{C}_{\mathrm{CTM}}(\mathbf{w}) = -\log_2 \mathrm{Dist}^T_{n,k}(\mathbf{w}) + O(1)$. For larger objects, however, exhaustive simulation is not tractable, since the number of machines grows exponentially with n and k.

Block Decomposition Method. To extend the CTM to larger objects, Zenil et al. (2018) introduced the Block Decomposition Method (BDM), which partitions the object into contiguous blocks and sums the complexity estimates of its smaller structures. Let $\mathbf{w} \in \Sigma^{H\times W}$ be a large object over a finite alphabet Σ, and let Π be a lossless partition operator such that

$$\Pi(\mathbf{w}) = (\mathrm{w}_1, \ldots, \mathrm{w}_m), \qquad \mathbf{w} = \mathrm{w}_1 \otimes \cdots \otimes \mathrm{w}_m, \tag{21}$$

where each $\mathrm{w}_j \in \Sigma^{h\times w}$ is a contiguous block of size $h \times w$, and $\otimes$ denotes the reconstruction of $\mathbf{w}$ by concatenating the ordered blocks. If a block appears several times, its complexity is accounted for only once, with a logarithmic term for its frequency; the idea is that repeated structures need not be described from scratch. Let $\Pi_u(\mathbf{w}) \subseteq \Pi(\mathbf{w})$ denote the set of distinct blocks in the partition, and let n_{w_j} denote the number of occurrences of $\mathrm{w}_j \in \mathbf{w}$. The KCS complexity is then computed as

$$\mathcal{C}_{\mathrm{BDM}}(\mathbf{w}) = \sum_{\mathrm{w}_j \in \Pi_u(\mathbf{w})} \mathcal{C}_{\mathrm{CTM}}(\mathrm{w}_j) + \log_2 n_{\mathrm{w}_j}, \tag{22}$$

where the logarithmic term accounts for the complexity of repeating a block. Hence, repeated structures have diminishing complexity beyond their first occurrences. For example, consider the objects $\mathbf{w}_1 = aa \otimes aa \otimes ab \otimes aa$, and $\mathbf{w}_2 = aa \otimes ab \otimes ba \otimes bb$. Assuming $\mathcal{C}_{\mathrm{CTM}}(aa) = \mathcal{C}_{\mathrm{CTM}}(ab) = \mathcal{C}_{\mathrm{CTM}}(ba) = \mathcal{C}_{\mathrm{CTM}}(bb) = 1$, then $\mathrm{BDM}(\mathbf{w}_1) = 1 + \log_2 3 + 1 \approx 3.6$ and $\mathrm{BDM}(\mathbf{w}_2) = 4$.

Limitations. In practice, the accuracy of $\mathcal{C}_{\mathrm{BDM}}$ is limited by the finite CTM tables it relies on, where values are computed from a restricted machine class $\mathcal{M}^T_{n,k}$ with fixed n, k, and T. As a result, differences in estimates may reflect either structural differences or limitations of the underlying CTM approximation. In particular, partition choices are not invariant in practice. Although they represent the same object up to an additive constant, different block shapes, e.g., (2×1) versus (1×2), produce different local blocks and multiplicities, and therefore different $\mathcal{C}_{\mathrm{BDM}}$ estimates. In the unbounded setting, such differences are bounded up to an additive constant for fixed lossless partition functions (Kolmogorov, 1965).

The most extensive CTM table is the 5-state, 2-symbol enumeration of Soler-Toscano et al. (2014), which considered over 2.6×10^{13} machines and required 450 CPU days. As the basis for the BDM approximation, Zenil et al. (2018) showed that such finite CTM values still provide meaningful local rankings of short patterns and capture simple algorithmic structure[4]. However, extending CTM tables is computationally infeasible, especially in high-dimensional settings such as neural networks with large parameter spaces and high-precision values. In practice, objects are therefore first transformed to finite representations, and then decomposed into structures that are closer to what the available CTM tables can support.

Compression Proxies. A common approach is to use compression algorithms as proxies for KCS complexity. The better an object compresses, the less complex we expect it to be. Such approximations have been used to reason about structure and how generative patterns emerge. For instance, GZIP (Gailly and Adler, 1992) has been used to approximate Gibbs entropy in molecular systems (Liu and Simine, 2024) and as a language-agnostic metric for distinguishing native from second-language writing (Alzahrani, 2024). Bzip2 (Seward, 1996) has been used as a proxy for KCS complexity to estimate evolutionary distance between SARS-CoV-2 sequences (Machado et al., 2021). In image settings, JPG and PNG have been used as practical proxies for categorizing paintings by style and artist (Rigau et al., 2007), and for distinguishing schools of orthodox iconography (Peptenatu et al., 2022). These methods are useful diagnostics, but they are not CTM or BDM estimators, since they rely on compressor-specific coding schemes rather than algorithmic probability tables.

[4]We refer to Zenil et al. (2018) for further practical considerations, including overlapping versus non-overlapping partitions and their effect on the estimates.

B QuBD Method Details

B.1 Quantization

Neural network weights are real-valued. For b-bit scalar quantization, we map $\mathbf{w} \in \mathbb{R}^d$ to integer codes $\mathbf{q} \in \{0, \dots, q_{\max}\}^d$, where $q_{\max} = 2^b - 1$, by applying a quantizer elementwise. Let

$$\mathrm{w}_{\min} := \min_i \mathrm{w}_i, \qquad \mathrm{w}_{\max} := \max_i \mathrm{w}_i, \qquad \Delta_{\mathrm{w}} := \mathrm{w}_{\max} - \mathrm{w}_{\min}, \qquad s := \frac{\Delta_{\mathrm{w}}}{q_{\max}}.$$

Define the (min–max) uniform affine quantizer $\mathbf{q} = Q_b(\mathbf{w})$, where $Q_b : \mathbb{R}^d \to \{0, \dots, q_{\max}\}^d$ by the elementwise operation:

$$q_i = \begin{cases} \mathrm{clip}\left(\left\lfloor \dfrac{\mathrm{w}_i - \mathrm{w}_{\min}}{s} \right\rceil, \ 0, \ q_{\max} \right), & \Delta_{\mathrm{w}} \geq \varepsilon, \\ 0, & \Delta_{\mathrm{w}} < \varepsilon, \end{cases} \tag{23}$$

where $\lfloor \cdot \rceil$ rounds to nearest integer, $\mathrm{clip}(\cdot, 0, q_{\max})$ clamps to the valid range, and $\varepsilon > 0$ is a small constant for numerical stability. The corresponding dequantizer $D_b : \{0, \dots, q_{\max}\}^d \to \mathbb{R}^d$ is

$$\widehat{\mathbf{w}} = D_b(\mathbf{q}) = s\,\mathbf{q} + \mathrm{w}_{\min}\mathbf{1}. \tag{24}$$

When performing affine quantization, we adhere to the standard practice and set $w_{\min}$ and $w_{\max}$ to the 0.05th and 99.95th percentiles of each weight tensor. That is, we keep the central 99.9% of values and clip the 0.1% most extreme values, which may include outliers.

B.2 Example of Bit-Plane Decomposition

Consider an object $\mathbf{w} = [\mathrm{w}_1, \mathrm{w}_2]$ with $\mathrm{w}_1 = 4$ and $\mathrm{w}_2 = 13$. And for intuition, let the target precision be $b^\star = 4$. Then the lossless quantized target is

$$\mathbf{w}^\star = Q_{b^\star}(\mathbf{w}) = (4, 13) = (0100_2, 1101_2). \tag{25}$$

The corresponding bit-planes are

$$\boldsymbol{\beta}_3^\star = (0, 1), \qquad \boldsymbol{\beta}_2^\star = (1, 1), \qquad \boldsymbol{\beta}_1^\star = (0, 0), \qquad \boldsymbol{\beta}_0^\star = (0, 1). \tag{26}$$

If we keep only $b = 2$ most significant bit-planes, then

$$\mathbf{q}_2 = \left\lfloor \frac{\mathbf{w}^\star}{2^{4-2}} \right\rfloor = \left\lfloor \frac{(4, 13)}{4} \right\rfloor = (1, 3) = (01_2, 11_2). \tag{27}$$

So only the first two bit-planes, $\boldsymbol{\beta}_3^\star$ and $\boldsymbol{\beta}_2^\star$, are retained. The omitted lower-bit residual is

$$\delta_2 = \mathbf{w}^\star - 2^{4-2} q_2 = (4, 13) - 4(1, 3) = (0, 1) = (00_2, 01_2). \tag{28}$$

Then we have

$$\mathbf{w}^\star = (4, 13) = (0100_2, 1101_2) = 2^{4-2}(01_2, 11_2) + (00_2, 01_2). \tag{29}$$

In this example, $\mathrm{w}_1 = 4$ is already determined by its two retained MSB bit-planes, while $\mathrm{w}_2 = 13$ still requires the lower residual bit-planes to recover the symbol. Thus estimating $\mathcal{C}(\mathbf{q}_2)$ ignores the information stored in δ_2.

B.3 Weight Tensor Preprocessing

Prior to computing the QuBD complexity of a network, each of its weight tensors must be represented as a 2D binary matrix suitable for BDM estimation. This requires two steps: reshaping the weight tensor into a 2D matrix, and extracting its bit-planes as described in Sec. 3. The reshaping strategy matters because BDM is sensitive to the spatial adjacency of elements, and different reshaping choices expose different local structures to the CTM table. BDM is, however, invariant to transposing a matrix, so its orientation does not matter.

Weight Reshaping. Complexity is computed over the weight tensors of linear and convolutional layers, while biases, normalization parameters, and other parameters such as positional embeddings are excluded. Fully connected layers have weight tensors of shape $[C_{\text{out}}, C_{\text{in}}]$ and are already 2D, requiring no reshaping. Tensors with more than two dimensions (e.g., 4D convolutional kernels of shape $[C_{\text{out}}, C_{\text{in}}, H, W]$) are flattened along all but the first dimension, yielding a matrix of shape $[C_{\text{out}}, C_{\text{in}} \cdot H \cdot W]$. Each row then corresponds to one filter, with the weights of each kernel kept contiguous, and a 1×1 convolution yields the same matrix as a fully connected layer. Matrices with fewer than 4 rows or columns after reshaping are excluded, as they cannot be partitioned into blocks of the size supported by the precomputed CTM tables used in this work ($\pi = 4 \times 4$; Talaga and Tsampourakis 2024). Matrices whose dimensions are not multiples of 4 are partitioned without padding: rows and columns that do not fill a complete block are ignored, which for the 100 models in Fig. 1 affects less than 0.1% of the weights overall and at most 1.4% in any single model.

Robustness to Reshaping. The flattening of convolutional weights described above is one of several possible conventions. Since BDM is sensitive to which weights are adjacent, we test whether this choice affects the QuBD complexity ratio by replacing it with four alternative reshapings of the same weights:

(i) *Input-channel rows*, $[C_{\text{in}}, C_{\text{out}} \cdot H \cdot W]$: each row contains the kernels of one input channel instead of one filter.

(ii) *Channels-last*, $[C_{\text{out}}, H \cdot W \cdot C_{\text{in}}]$: the input channel varies fastest within a row, so that the same kernel position of different input channels is adjacent, and the weights of a single kernel are spread C_{in} entries apart.

(iii) *Closest-to-square*: the flattened weights are reshaped into the most square matrix $[r, c]$ with $r \cdot c = C_{\text{out}} \cdot C_{\text{in}} \cdot H \cdot W$, ignoring the semantics of the axes.

(iv) *Kernel mosaic*, $[C_{\text{out}} \cdot H, C_{\text{in}} \cdot W]$: each kernel is kept as its $H \times W$ patch, arranged in a grid of filters and input channels.

Quantization depends only on the weight values, so all reshapings yield identical quantized values and differ only in their arrangement. Fully connected layers are unaffected by the reshapings. We compute the model-level QuBD complexity ratio $\Delta\mathcal{C}_{\text{QuBD}}$ per bit-plane for the five models of Fig. 8 (8-bit quantization, clipping at the 99.9th percentile) and report the largest absolute deviation from the default over all bit-planes in Tab. 2.

Table 2: Largest absolute deviation (percentage points) of $\Delta\mathcal{C}_{\text{QuBD}}$ from the default reshaping over all 8 bit-planes. Dash denotes that the reshaping leaves every layer of the model unchanged.

Reshaping	ResNet-18	ResNet-50	ViT-B/16	EfficientNet-B0	MobileNetV3
(i) Input-channel rows	0.2	0.1	–	–	–
(ii) Channels-last	12.5	6.7	0.2	0.0	0.0
(iii) Closest-to-square	0.4	0.1	–	0.3	0.1
(iv) Kernel mosaic	1.6	0.8	0.2	0.5	0.1

Reshapings that keep the weights of each kernel together, (i), (iii), and (iv), change $\Delta\mathcal{C}_{\text{QuBD}}$ by at most 1.6 percentage points across all models and bit-planes. The only reshaping with a notable effect is channels-last (ii), which is also the only one that separates the weights of a kernel: in ResNet-18 and ResNet-50 it increases $\Delta\mathcal{C}_{\text{QuBD}}$ on the most significant bit-planes by up to 12.5 and 6.7 percentage points, respectively. This indicates that the structure captured by QuBD in convolutional layers lies within the spatial dimensions of the kernels, which the default reshaping preserves. For ViT-B/16, EfficientNet-B0 and MobileNetV3, the choice of reshaping has almost no effect, as nearly all of their weights lie in fully connected, 1×1 or depthwise layers, where (i), (ii) and (iv) coincide with the default and (iii) changes $\Delta\mathcal{C}_{\text{QuBD}}$ by at most 0.3 percentage points.

C QuBD Complexity

C.1 Information Identities for KCS Complexity

We list here some useful KCS complexity identities and briefly interpret them to build further intuition. We refer to Li et al. (2008) for additional details.

Algorithmic Mutual Information. For finite binary strings $x, y, z \in \{0,1\}^*$, the algorithmic information in x about y (and not) conditioned on z is

$$I(x : y) := \mathcal{C}(y) - \mathcal{C}(y \mid x), \quad I(x : y \mid z) := \mathcal{C}(y \mid z) - \mathcal{C}(y \mid x, z). \tag{30}$$

Chain Rule for KCS Complexity. For finite binary strings $x, y \in \{0,1\}^*$,

$$\begin{aligned} \mathcal{C}(x, y) &= \mathcal{C}(x) + \mathcal{C}\big(y \mid x, \mathcal{C}(x)\big) + O(1) \\ &\equiv \mathcal{C}(x) + \mathcal{C}\big(y \mid x, \mathcal{C}(x)\big) = \mathcal{C}(y) + \mathcal{C}\big(x \mid y, \mathcal{C}(y)\big) + O(1). \end{aligned} \tag{31}$$

Symmetry of Information. For finite binary strings $x, y \in \{0,1\}^*$,

$$\begin{aligned} I\big((x, \mathcal{C}(x)) : y\big) &= I\big((y, \mathcal{C}(y)) : x\big) + O(1) \\ &\equiv \mathcal{C}(y) - \mathcal{C}\big(y \mid x, \mathcal{C}(x)\big) = \mathcal{C}(x) - \mathcal{C}\big(x \mid y, \mathcal{C}(y)\big) + O(1). \end{aligned} \tag{32}$$

Eq. (30) isolates the part of y that can be reused from x, and the residual $\mathcal{C}(y \mid x)$, which remains irreducible. But note that $I(x : y) \neq I(y : x)$ in general, because the description of one object inside another is not self-delimiting, i.e., from x alone we cannot know where the description of x ends inside a description of the pair (x, y). This is apparent in Eq. (31) where the additional conditioning on $\mathcal{C}(x)$ in $\mathcal{C}(y \mid x, \mathcal{C}(x))$ provides the information to distinguish the objects. Conveniently, Eq. (32) shows exactly when mutual information between objects is in fact symmetric.

However, it can be clearer to reason about the symmetry of information assuming *composite* objects, e.g., (x, y) are self-delimiting. We therefore omit additional conditionals in our analysis, which instead incurs an error in information of $O(\log b^\star d)$ for objects with bit length $O(b^\star d)$.

C.2 Proof of Proposition 3.1 (Residual Decomposition)

By construction $\mathbf{w}^\star = 2^{(b^\star - b)}\mathbf{q}_b + \delta_b$, so the target object $\mathbf{w}^\star$ and the composite object $(\mathbf{q}_b, \delta_b)$ describe each other up to a fixed additive constant:

$$\mathcal{C}(\mathbf{w}^\star) = \mathcal{C}(\mathbf{q}_b, \delta_b) + O(1). \tag{33}$$

Applying the chain rule from Eq. (31) to $(\mathbf{q}_b, \delta_b)$ gives

$$\mathcal{C}(\mathbf{q}_b, \delta_b) = \mathcal{C}(\mathbf{q}_b) + \mathcal{C}(\delta_b \mid \mathbf{q}_b, \mathcal{C}(\mathbf{q}_b)) + O(1). \tag{34}$$

Since the objects here have bit length $O(b^\star d)$, the components are self-delimiting up to a logarithmic term, and therefore

$$\begin{aligned} &\mathcal{C}(\mathbf{q}_b, \delta_b) = \mathcal{C}(\mathbf{q}_b) + \mathcal{C}(\delta_b \mid \mathbf{q}_b) + O(\log(b^\star d)) \\ \implies \quad &\mathcal{C}(\mathbf{w}^\star) = \mathcal{C}(\mathbf{q}_b) + \mathcal{C}(\delta_b \mid \mathbf{q}_b) + O(\log(b^\star d)). \end{aligned} \tag{35}$$

Letting $R_b = \mathcal{C}(\delta_b \mid \mathbf{q}_b)$ gives Eq. (9). That is, if $\mathbf{w}^\star$ is estimated only from the retained prefix $\mathbf{q}_b$, then the estimate misses the residual information contained in the discarded lower bits, up to the logarithmic term.

It remains to bound the size of R_b. By the construction, each residual element $\delta_{b,i}$ can take one of $2^{(b^\star - b)}$ possible values, namely from $\{0, \ldots, 2^{(b^\star - b)} - 1\}$. Therefore, it requires at most $b^\star - b$ bits to encode. Over all d elements, the bit length of δ_b is at most $(b^\star - b)d$. A conditional description of δ_b given $\mathbf{q}_b$ therefore cannot be longer than this bit length up to an additive constant, so we get

$$\begin{aligned} &0 \leq \mathcal{C}(\delta_b \mid \mathbf{q}_b) \leq (b^\star - b)d + O(1) \\ \implies \quad &0 \leq R_b \leq (b^\star - b)d + O(1). \end{aligned} \tag{36}$$

Thus, the residual estimation gap is bounded above in the number of discarded bits, $O((b^\star - b)d)$. Finally, if $b = b^\star$, then no lower bit-planes are omitted, so $\delta_b = 0$ and $R_b = \mathcal{C}(0 \mid \mathbf{q}_b) = O(1)$. □

C.3 Proof of Proposition 3.2 (Bit-Plane Residual Loss)

By definition of the MSB-prefixes, for each $i \in \{1, \dots, d\}$,

$$\mathbf{q}_{b+1,i} = \left\lfloor \frac{\mathrm{w}_i^\star}{2^{(b^\star - b)-1}} \right\rfloor = 2 \left\lfloor \frac{\mathrm{w}_i^\star}{2^{(b^\star - b)}} \right\rfloor + \beta^\star_{i,(b^\star-b)-1} = 2\mathbf{q}_{b,i} + \beta^\star_{i,(b^\star-b)-1}, \tag{37}$$

which can be more clearly expressed by

$$\mathbf{q}_{b+1} = 2\mathbf{q}_b + \boldsymbol{P}_{b+1}. \tag{38}$$

Now we write the residual decomposition at precisions b and $b+1$:

$$\mathbf{w}^\star = 2^{(b^\star-b)}\mathbf{q}_b + \delta_b = 2^{(b^\star-b)-1}\mathbf{q}_{b+1} + \delta_{b+1}. \tag{39}$$

Substituting $\mathbf{q}_{b+1} = 2\mathbf{q}_b + \boldsymbol{P}_{b+1}$ yields

$$\begin{aligned} 2^{(b^\star-b)}\mathbf{q}_b + \delta_b &= 2^{(b^\star-b)-1}(2\mathbf{q}_b + \boldsymbol{P}_{b+1}) + \delta_{b+1} \\ \implies \quad \delta_b &= 2^{(b^\star-b)-1}\boldsymbol{P}_{b+1} + \delta_{b+1}. \end{aligned} \tag{40}$$

So, once $\mathbf{q}_b$ is fixed, the residual δ_b and the pair $(\boldsymbol{P}_{b+1}, \delta_{b+1})$ describe each other up to an additive constant. Hence,

$$\mathcal{C}(\delta_b \mid \mathbf{q}_b) = \mathcal{C}(\boldsymbol{P}_{b+1}, \delta_{b+1} \mid \mathbf{q}_b) + O(1). \tag{41}$$

Applying the chain rule and absorbing the self-delimiting overhead into $O(\log(b^\star d))$, gives

$$\begin{aligned} \mathcal{C}(\delta_b \mid \mathbf{q}_b) &= \mathcal{C}(\boldsymbol{P}_{b+1} \mid \mathbf{q}_b) + \mathcal{C}(\delta_{b+1} \mid \mathbf{q}_b, \boldsymbol{P}_{b+1}) + O(\log(b^\star d)) \\ &= \mathcal{C}(\boldsymbol{P}_{b+1} \mid \mathbf{q}_b) + \mathcal{C}(\delta_{b+1} \mid \mathbf{q}_{b+1}) + O(\log(b^\star d)). \end{aligned} \tag{42}$$

The last equality holds because $\mathbf{q}_{b+1}$ and $(\mathbf{q}_b, \boldsymbol{P}_{b+1})$ describe each other by a fixed rule, namely

$$\mathbf{q}_{b+1} = 2\mathbf{q}_b + \boldsymbol{P}_{b+1}, \qquad \mathbf{q}_b = \left\lfloor \frac{\mathbf{q}_{b+1}}{2} \right\rfloor, \qquad \boldsymbol{P}_{b+1} = \mathbf{q}_{b+1} \bmod 2. \tag{43}$$

So replacing one condition with the other changes the conditional complexity by at most $O(1)$.

Using $R_b = \mathcal{C}(\delta_q \mid \mathbf{q}_b)$ and $R_{b+1} = \mathcal{C}(\delta_{b+1} \mid \mathbf{q}_{b+1})$,

$$\begin{aligned} R_b &= \mathcal{C}(\boldsymbol{P}_{b+1} \mid \mathbf{q}_b) + R_{b+1} + O(\log(b^\star d)) \\ \implies \quad R_b - R_{b+1} &= \mathcal{C}(\boldsymbol{P}_{b+1} \mid \mathbf{q}_b) + O(\log(b^\star d)). \end{aligned} \tag{44}$$

Since $\mathcal{C}(\boldsymbol{P}_{b+1} \mid \mathbf{q}_b) \geq 0$, we get

$$R_b \geq R_{b+1} - O(\log(b^\star d)). \tag{45}$$

Thus, the residual loss is non-increasing up to the same logarithmic term. If the new bit-plane is not already described by $\mathbf{q}_b$ and contributes information beyond that term, then $R_b > R_{b+1}$. □

C.3.1 Cumulative Bit-Plane Refinement

Corollary C.1 (Cumulative Bit-Plane Refinement). *For $1 \leq b < r \leq b^\star$, let $R_s := \mathcal{C}(\delta_s \mid \mathbf{q}_s)$ and $s \in \{b, r\}$. Then we obtain*

$$\begin{aligned} R_b &= \mathcal{C}(\mathbf{q}_r \mid \mathbf{q}_b) + R_r + O(\log(b^\star d)) \\ \implies \quad \mathcal{C}(\mathbf{q}_r) - \mathcal{C}(\mathbf{q}_b) &= R_b - R_r + O(\log(b^\star d)). \end{aligned} \tag{46}$$

In particular, setting $r = b^\star$ gives

$$R_b = \mathcal{C}(\mathbf{w}^\star \mid \mathbf{q}_b) + O(\log(b^\star d)), \qquad R_{b^\star} = O(1), \tag{47}$$

since $\mathbf{q}_{b^\star} = \mathbf{w}^\star$ and $\delta_{b^\star} = 0$. So adding bit-planes from precision b to precision r removes the information needed to refine $\mathbf{q}_b$ into $\mathbf{q}_r$, up to the same logarithmic term. At full precision, no residual information remains beyond the additive constant.

Proof. From Prop. 3.1, applied at precisions b and r,

$$\mathcal{C}(\mathbf{w}^\star) = \mathcal{C}(\mathbf{q}_b) + R_b + O(\log(b^\star d)), \qquad \mathcal{C}(\mathbf{w}^\star) = \mathcal{C}(\mathbf{q}_r) + R_r + O(\log(b^\star d)). \tag{48}$$

Subtracting gives

$$\mathcal{C}(\mathbf{q}_r) - \mathcal{C}(\mathbf{q}_b) = R_b - R_r + O(\log(b^\star d)). \tag{49}$$

Now $\mathbf{q}_b$ is a fixed truncation of $\mathbf{q}_r$, so $\mathcal{C}(\mathbf{q}_b \mid \mathbf{q}_r) = O(1)$, and hence

$$\mathcal{C}(\mathbf{q}_b, \mathbf{q}_r) = \mathcal{C}(\mathbf{q}_r) + O(1). \tag{50}$$

Applying the chain rule to $(\mathbf{q}_b, \mathbf{q}_r)$ therefore gives

$$\begin{aligned} & \mathcal{C}(\mathbf{q}_r) = \mathcal{C}(\mathbf{q}_b) + \mathcal{C}(\mathbf{q}_r \mid \mathbf{q}_b) + O(\log(b^\star d)) \\ \Longrightarrow \quad & \mathcal{C}(\mathbf{q}_r) - \mathcal{C}(\mathbf{q}_b) = \mathcal{C}(\mathbf{q}_r \mid \mathbf{q}_b) + O(\log(b^\star d)). \end{aligned} \tag{51}$$

Comparing the two expressions for $\mathcal{C}(\mathbf{q}_r) - \mathcal{C}(\mathbf{q}_b)$ yields

$$\begin{aligned} & R_b - R_r = \mathcal{C}(\mathbf{q}_r \mid \mathbf{q}_b) + O(\log(b^\star d)) \\ \Longrightarrow \quad & R_b = \mathcal{C}(\mathbf{q}_r \mid \mathbf{q}_b) + R_r + O(\log(b^\star d)). \end{aligned} \tag{52}$$

Finally, set $r = b^\star$. Then $\mathbf{q}_{b^\star} = \mathbf{w}^\star$ and $\delta_{b^\star} = 0$, so

$$\begin{aligned} & R_{b^\star} = \mathcal{C}(\delta_{b^\star} \mid \mathbf{q}_{b^\star}) = O(1) \\ \Longrightarrow \quad & R_b = \mathcal{C}(\mathbf{w}^\star \mid \mathbf{q}_b) + O(\log(b^\star d)). \end{aligned} \tag{53}$$

Therefore, the residual loss at precision b is the information needed to refine $\mathbf{q}_b$ into the target object $\mathbf{w}^\star$, up to the same logarithmic term. As more bit-planes are retained, this missing information decreases; at full precision only the additive constant remains.

Repeated application of Prop. 3.2 gives the equivalent telescoping sum

$$R_b - R_r = \sum_{j=b}^{r-1} \mathcal{C}(\boldsymbol{P}_{j+1} \mid \mathbf{q}_j) + O((r-b)\log(b^\star d)), \tag{54}$$

which shows that one conditional contribution is removed at each added bit-plane. □

C.4 Empirical Validation of the Decrease in Residual Loss

We test the estimation gap using the CTM table as a finite reference. By the coding theorem, CTM assigns low complexity to blocks with high empirical algorithmic probability, up to an additive constant (Solomonoff, 1964; Levin, 1974; Delahaye and Zenil, 2012). For each block size π, we

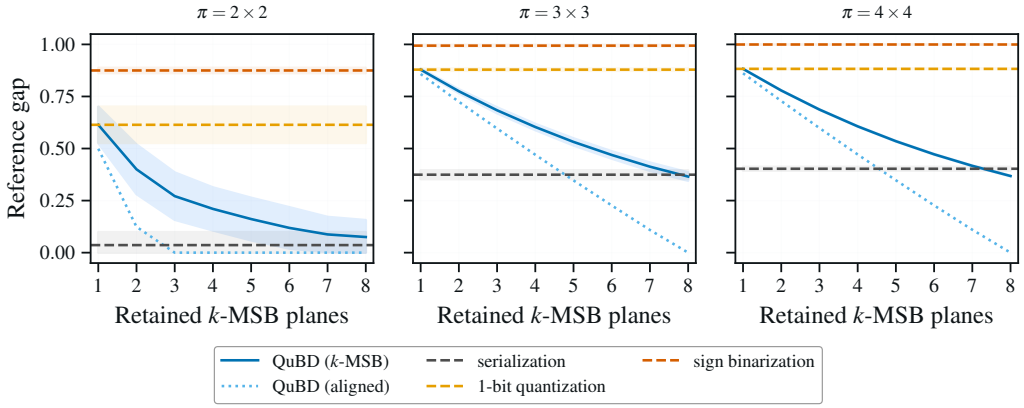


Figure 9: Relative gap (±1 std.) to the finite CTM-support reference over 100 trials (fixed symbol relabelings). The aligned QuBD control decreases as more MSB bit-planes are retained, validating the residual refinement predicted by Prop. 3.2. Under symbol relabeling, QuBD remains closer to the reference than serialization, one-bit quantization, and sign binarization.

sum the CTM values of all supported blocks once. This gives a reference target object for the local structures available to the finite estimator, without BDM multiplicity terms.

We construct an 8-bit object whose bit-planes expose all supported CTM blocks. For intuition, consider a binary CTM table with 1×2 blocks and support $\{00, 01, 10, 11\}$. With $b = 2$, choose $\boldsymbol{\beta}_1 = 0011$ and $\boldsymbol{\beta}_0 = 0101$, which gives the quantized object $[0, 1, 2, 3]$. Under QuBD, $\Pi(\boldsymbol{\beta}_1) = \{00, 11\}$ and $\Pi(\boldsymbol{\beta}_0) = \{01, 01\}$, so the exposed support is $\{00, 01, 11\}$. The reverse construction, if the binary string is $00011011 = 00\,01\,10\,11$, then serialization exposes all four supported blocks.

For the aligned object, we note that the gap decreases as more bit-planes are retained, matching Prop. 3.2 and Cor. C.1. For the comparison, we keep the same target object and apply a random symbol relabeling before exposure. Each relabeling assigns a unique new value to each value in $\{0, \ldots, 255\}$. This preserves the target object up to an additive constant, while changing the binary blocks exposed to the CTM table. Fig. 9 shows that QuBD remains closer to the finite reference than serialization, one-bit quantization, and sign binarization under these relabelings.

D Saturation in Finite-state Machines

D.1 Additional Saturation Details

As discussed in Sec. 3, lossless representations can still expose objects differently. Besides low residual loss, a representation should expose substructures that remain informative under the finite support of the CTM table, i.e., the set of structures available to the estimator.

Fix a block size π and a finite CTM table with supported states S_π. Let $f(u)$ denote the empirical frequency of a block $u \in S_\pi$, similar to Eq. (20). We write $z_{b,r} = \phi(\mathbf{q}_b)$ to denote the exposure of $\mathbf{w}$ with b-bit quantization and representation r. Partition $z_{b,r}$ into blocks by $\Pi_\pi(z_{b,r}) = (\mathrm{w}_1, \ldots, \mathrm{w}_m)$, where $m = |z_{b,r}|/\pi$. For each supported block $u \in S_\pi$, define

$$\begin{aligned} c_\pi(u; z_{b,r}) &:= |\{i \le m : \mathrm{w}_i = u\}|, \\ A_\pi(z_{b,r}) &:= \{u \in S_\pi : c_\pi(u; z_{b,r}) > 0\}, \\ a_\pi(z_{b,r}) &:= |A_\pi(z_{b,r})|. \end{aligned} \tag{55}$$

Here $c_\pi(u; z_{b,r})$ counts how often u is exposed under $z_{b,r}$, while $a_\pi(z_{b,r})$ is the number of supported blocks that appear. The KCS estimate of Eq. (2) can then be expressed as

$$\mathcal{C}_{\mathrm{BDM}}(z_{b,r}) = \underbrace{\sum_{u \in A_\pi(z_{b,r})} -\log_2 f(u)}_{\text{new blocks}} + \underbrace{\sum_{u \in A_\pi(z_{b,r})} \log_2 c_\pi(u; z_{b,r})}_{\text{multiplicity}}. \tag{56}$$

Importantly, after most supported blocks have appeared, the first term changes little, and the estimate grows mainly through multiplicities. In this case, we consider CTM tables in the finite setting saturated.

To compare exposures, let $S^{(b)}_{\pi,r} \subseteq S_\pi$ be the blocks that can be exposed by $z_{b,r}$, and let $H^{(b)}_{\pi,r} := |S^{(b)}_{\pi,r}|$. A CTM table may support many structures, but a given exposure may only reach some of them. Let $\rho_{\pi,r}(u)$ denote the probability that one exposed block of $z_{b,r}$ equals u. Then, for m exposed blocks, we have

$$\mathbb{E}[a_\pi(z_{b,r})] = \sum_{u \in S^{(b)}_{\pi,r}} \left[1 - (1 - \rho_{\pi,r}(u))^m\right], \quad \sigma_{\pi,r}(m) := \frac{\mathbb{E}[a_\pi(z_{b,r})]}{H^{(b)}_{\pi,r}}. \tag{57}$$

Here $\sigma_{\pi,r}(m)$ is the fraction of reachable support expected to appear after m exposed blocks. We call the estimator $(1-\epsilon)$-saturated when $\sigma_{\pi,r}(m) \ge 1 - \epsilon$.

D.2 Proof of Proposition 3.3 (Finite-table Saturation)

Fix b, r, π, and $S^{(b)}_{\pi,r}$. We omit the fixed b for readability and write $S_{\pi,r}$ with $H_{\pi,r} := |S_{\pi,r}|$. Let $\Pi_\pi(z_{b,r}) = (\mathrm{w}_1, \ldots, \mathrm{w}_m)$ be the exposed blocks of size π.

For each $u \in S_{\pi,r}$, define $I_u := \mathbf{1}\{c_\pi(u; z_{b,r}) > 0\}$. Then $a_\pi(z_{b,r}) = \sum_{u \in S_{\pi,r}} I_u$, and by linearity

$$\mathbb{E}[a_\pi(z_{b,r})] = \mathbb{E}\left[\sum_{u \in S_{\pi,r}} I_u\right] = \sum_{u \in S_{\pi,r}} \mathbb{E}[I_u] = \sum_{u \in S_{\pi,r}} \mathbb{P}(c_\pi(u; z_{b,r}) > 0). \tag{58}$$

Let $\rho_{\pi,r}(u)$ be the probability that one exposed block equals u. Since the exposed blocks are sampled independently, the probability that none of the m exposed blocks equals u is $(1 - \rho_{\pi,r}(u))^m$. Hence

$$\mathbb{E}[a_\pi(z_{b,r})] = \sum_{u \in S_{\pi,r}} \left[1 - (1 - \rho_{\pi,r}(u))^m\right]. \tag{59}$$

In the uniform setting, which serves as a worst-case upper bound on the number of blocks required for saturation, we assume

$\rho_{\pi,r}(u) = 1/H_{\pi,r}$ for all $u \in S_{\pi,r}$, and therefore

$$\begin{aligned} \mathbb{E}[a_\pi(z_{b,r})] &= \sum_{u \in S_{\pi,r}} \left[1 - \left(1 - \frac{1}{H_{\pi,r}}\right)^m\right] \\ &= H_{\pi,r}\left[1 - \left(1 - \frac{1}{H_{\pi,r}}\right)^m\right] \approx H_{\pi,r}\left(1 - e^{-m/H_{\pi,r}}\right), \end{aligned} \tag{60}$$

where the last step uses $(1 - x)^m \approx e^{-mx}$ for small x. By definition, $\sigma_{\pi,r}(m) = \frac{\mathbb{E}[a_\pi(z_{b,r})]}{H_{\pi,r}}$, so

$$\begin{aligned} \sigma_{\pi,r}(m) &= \frac{H_{\pi,r}\left[1 - \left(1 - \frac{1}{H_{\pi,r}}\right)^m\right]}{H_{\pi,r}} \\ &= 1 - \left(1 - \frac{1}{H_{\pi,r}}\right)^m \approx 1 - e^{-m/H_{\pi,r}}. \end{aligned} \tag{61}$$

Thus, saturation is controlled by $m/H_{\pi,r}$. We now write $a_\pi(m)$ for the number of distinct supported blocks seen for an exposure of m blocks. Then the expected number of new supported blocks added by one more exposed block is

$$\begin{aligned} \mathbb{E}[a_\pi(m+1)] - \mathbb{E}[a_\pi(m)] &= H_{\pi,r}\left[\left(1 - \frac{1}{H_{\pi,r}}\right)^m - \left(1 - \frac{1}{H_{\pi,r}}\right)^{m+1}\right] \\ &= \left(1 - \frac{1}{H_{\pi,r}}\right)^m \approx e^{-m/H_{\pi,r}}. \end{aligned} \tag{62}$$

Thus, after m blocks pass the saturation threshold, additional exposed blocks mostly repeat blocks already present in the finite table, and the estimate grows through multiplicity.

Finally, $(1 - \epsilon)$-saturation means $\sigma_{\pi,r}(m) \geq 1 - \epsilon$, so

$$\begin{aligned} \sigma_{\pi,r}(m) \geq 1 - \epsilon &\iff 1 - \left(1 - \frac{1}{H_{\pi,r}}\right)^m \geq 1 - \epsilon \\ &\iff \left(1 - \frac{1}{H_{\pi,r}}\right)^m \leq \epsilon \\ &\iff m \log\left(1 - \frac{1}{H_{\pi,r}}\right) \leq \log \epsilon \\ &\iff m \geq \frac{\log \epsilon}{\log(1 - 1/H_{\pi,r})} \approx H_{\pi,r} \log \frac{1}{\epsilon}, \end{aligned} \tag{63}$$

where the inequality flips in the last step since $\log(1 - 1/H_{\pi,r}) < 0$, and the approximation uses $\log(1 - x) \approx -x$.

After saturation, the finite table mostly contributes through repeated counts. For uniform exposure, the expected count of each supported block is $m/H_{\pi,r}$, and for large m these counts concentrate around this value. And when $a_\pi(z_{b,r}) \approx H_{\pi,r}$,

$$\sum_{u \in A_\pi(z_{b,r})} \log_2 c_\pi(u; z_{b,r}) \approx \sum_{u \in S_{\pi,r}} \log_2\left(\frac{m}{H_{\pi,r}}\right) = H_{\pi,r} \log_2\left(\frac{m}{H_{\pi,r}}\right). \tag{64}$$

Thus, after saturation, further growth in the estimate comes mainly from multiplicity terms rather than newly observed blocks. □

D.3 Saturation Across Exposures

We compare binarization, serialization, and bit-plane decomposed exposures by counting how many binary blocks each representation exposes in the CTM table. Let d be the number of quantized symbols in $\mathbf{q}_b \in \Sigma_b^d$, and let π be the binary block size. Since the available CTM tables are two-symbol tables, every exposure is evaluated through binary blocks in $S_\pi \subseteq \{0,1\}^\pi$. The relevant quantity is the number of exposed blocks m_r relative to the reachable support $H_{\pi,r}^{(b)}$, and not only d, the number of b-ary symbols alone.

For the three exposures, the number of blocks is

$$m_{\text{bin}} = \frac{d}{\pi}, \qquad m_{\text{ser}} = \frac{bd}{\pi}, \qquad m_{\text{plane},j} = \frac{d}{\pi}, \quad j = 1, \dots, k, \tag{65}$$

where $m_{\text{plane},j}$ is counted per retained bit-plane. If k bit-planes are retained, the estimator considers k binary objects separately and independently, and the aggregate reachable support is

$$H_{\pi,\text{plane}}^{(k)} = \sum_{j=1}^{k} H_{\pi,\text{plane},j} = kH_{\pi,\text{plane}}. \tag{66}$$

By Prop. 3.3, the object sizes needed before $(1-\epsilon)$-saturation are

$$\begin{aligned} d_{\text{bin}} &\gtrsim \pi H_{\pi,\text{bin}} \log \frac{1}{\epsilon}, \qquad d_{\text{ser}} \gtrsim \frac{\pi H_{\pi,\text{ser}}^{(b)}}{b} \log \frac{1}{\epsilon}, \\ d_{\text{plane},j} &\gtrsim \pi H_{\pi,\text{plane},j} \log \frac{1}{\epsilon}, \quad \text{and} \quad d_{\text{plane}}^{(k)} \gtrsim k\pi H_{\pi,\text{plane}}^{(k)} \log \frac{1}{\epsilon}. \end{aligned} \tag{67}$$

Binarization exposes only d/π blocks, but estimates a one-bit target. Serialization preserves the b-bit target, but exposes bd/π blocks to one finite table and therefore saturates earlier. Bit-plane exposure preserves the b-bit target and scores the planes separately. We note that increasing k reduces residual loss and increases the aggregate support available to the estimator, while each plane is still exposed at the slower rate per bit-plane. For a concrete example, take $b = 8$ and $\pi = 16$, corresponding to binary 4×4 blocks. The CTM support is of size $|H| = 2^{16}$. With $\epsilon = 0.1$, we have

$$H \log \frac{1}{\epsilon} = 2^{16} \log \frac{1}{0.1} \approx 1.51 \times 10^5. \tag{68}$$

The corresponding object scales are

$$\begin{aligned} d_{\text{ser}} &\gtrsim \frac{\pi H}{b} \log \frac{1}{\epsilon} = \frac{16}{8} \cdot 1.51 \times 10^5 \approx 3.0 \times 10^5, \\ d_{\text{bin}} &\gtrsim \pi H \log \frac{1}{\epsilon} = 16 \cdot 1.51 \times 10^5 \approx 2.4 \times 10^6, \\ d_{\text{plane}} &\gtrsim \pi H \log \frac{1}{\epsilon} = 16 \cdot 1.51 \times 10^5 \approx 2.4 \times 10^6. \end{aligned} \tag{69}$$

Serialization reaches the same saturation about $b = 8$ times earlier than one bit-plane. Binarization has the same saturation scale as a single bit-plane in this example, but it estimates a different one-bit target and discards the lower-bit residuals. We include it merely for completeness. So bit-plane exposure keeps the b-bit target while delaying saturation in the number of bit-planes. The delayed saturation rates across the different exposures are empirically observed in Fig. 10. Importantly, the estimation gap for increasing object sizes caused by the different representations, and when they fail, as discussed in Sec. 3.2, is seen in Figs. 11 and 12. This shows how the delayed saturation yields more reliable estimates for larger objects.

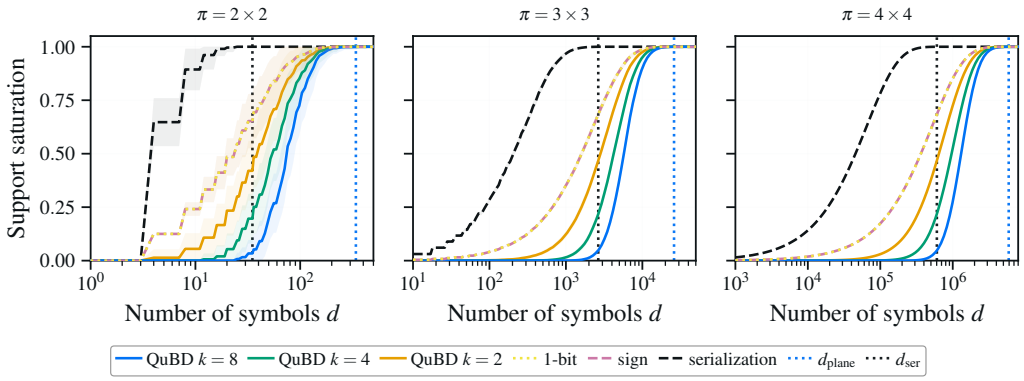


Figure 10: Empirical saturation rates as the number of symbols d exposed to the CTM table increases across block sizes $\pi \in \{2 \times 2, 3 \times 3, 4 \times 4\}$. Each curve shows the fraction of available support observed for each representation. Vertical lines denote the theoretical saturation rates derived for bit-plane decomposition in QuBD (d_{plane}) and serialized (d_{ser}) representations. Symbols are sampled uniformly at random from the support of the CTM-table for up to 10^6 draws. Results are averaged over 100 trials.

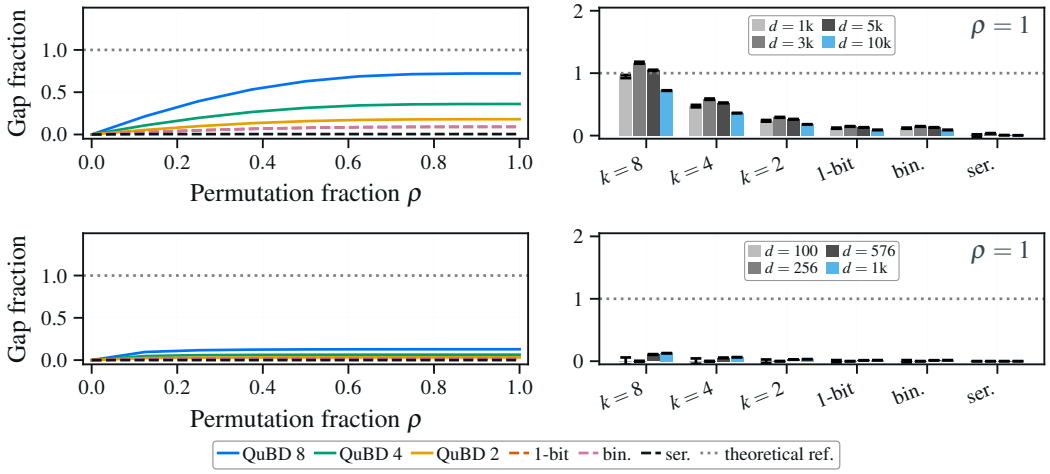


Figure 11: Complexity gap with **(top)** $\pi = (3 \times 3)$ and **(bottom)** $\pi = (2 \times 2)$, between estimations of an ordered 8-ary object and its permutation, where ρ is the fraction of symbols d permuted. The gap is shown **(left)** for the largest object in consideration, $d \in \{1\text{k}, 10\text{k}\}$ symbols, as ρ increases, and **(right)** across object sizes for different methods when $\rho = 1$. QuBD keeps a larger and more stable estimation gap than serialization, one-bit, and sign-binarization. Complexity values are normalized by the theoretical reference for its type class (Cover, 1999). Each object is sampled uniformly at random, and results are averaged over 100 trials.

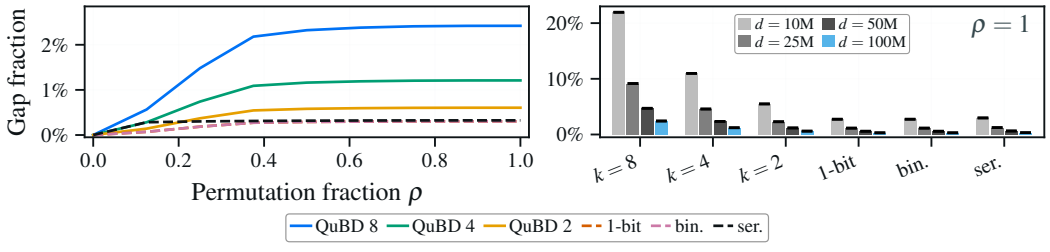


Figure 12: As Fig. 11 for $\pi = (4 \times 4)$ with normalized $(\%)$ gap shown **(left)** for an object with $d = 100$M symbols and **(right)** across object sizes.

E Additional Experimental Details

The accuracy-versus-complexity, learning curve, and grokking experiments were performed on a desktop workstation with an Intel i7 processor, 64 GB memory, and an Nvidia RTX 5090 GPU with 32 GB memory. The QuBD complexity across bit-planes, per-layer, and PTQ experiments were conducted on an Apple Silicon Mac, with the latter using the MPS backend. All code was implemented in PyTorch (Paszke et al., 2019). Specific choices for individual experiments are presented next.

E.1 Complexity Across Models and Bit-Planes Experiment

The results reported in Fig. 1 and Fig. 14 are based on 100 pretrained models sourced from the `timm` library (Wightman, 2019), spanning convolutional, transformer, and hybrid architectures. Each model is loaded with its default pretrained weights and `timm`'s default random initialization under a fixed seed. Following the weight tensor preprocessing described in Appx. B.3, each weight tensor is quantized to 8 bits with a uniform quantizer, clipping at the 99.9th percentile. This experiment requires no dataset evaluation and took approximately 80 minutes to complete on the reported hardware.

The 100 models, ordered by number of parameters, are: `edgenext_xx_small` (1.3M), `regnetx_002` (2.7M), `starnet_s1` (2.9M), `rexnet_100` (4.8M), `convnext_femto` (5.2M), `efficientnet_es_pruned` (5.4M), `tf_efficientnet_es` (5.4M), `deit_tiny_distilled_patch16_224` (5.9M), `hgnetv2_b0` (6.0M), `tf_mixnet_l` (7.3M), `nf_regnet_b1` (10.2M), `resnext26ts` (10.3M), `eca_resnext26ts` (10.3M), `pit_xs_224` (10.6M), `mobilenetv4_hybrid_medium` (11.1M), `shvit_s3` (14.2M), `vit_pwee_patch16_reg1_gap_256` (15.3M), `convnext_nano` (15.6M), `resnet26` (16.0M), `rexnet_200` (16.4M), `cspresnext50` (20.6M), `dla60_res2net` (20.8M), `gernet_m` (21.1M), `resnet50_clip_gap` (23.5M), `efficientvit_b2` (24.3M), `resnetaa50d` (25.6M), `resnet50d` (25.6M), `dla102x` (26.3M), `swin_s3_tiny_224` (28.3M), `focalnet_tiny_lrf` (28.6M), `poolformerv2_s36` (30.8M), `maxvit_tiny_tf_384` (31.0M), `fasternet_s` (31.2M), `selecsls42b` (32.5M), `sam2_hiera_small` (33.9M), `cs3darknet_x` (35.0M), `fastvit_mci2` (35.8M), `resnet61q` (36.8M), `levit_conv_384` (39.1M), `levit_384` (39.1M), `eca_nfnet_l1` (41.4M), `coatnet_rmlp_1_rw_224` (41.7M), `xception71` (42.3M), `crossvit_18_dagger_240` (44.3M), `crossvit_18_dagger_408` (44.6M), `efficientnet_h_b5` (45.9M), `xcit_small_24_p8_384` (47.6M), `legacy_seresnet101` (49.3M), `inception_next_small` (49.4M), `swinv2_cr_small_ns_224` (49.7M), `cs3se_edgenet_x` (50.7M), `gcvit_small` (51.1M), `dla169` (53.4M), `tf_efficientnetv2_m` (54.1M), `regnetx_160` (54.3M), `tresnet_l` (56.0M), `convformer_m36` (57.1M), `repvgg_b1` (57.4M), `regnetz_e8` (57.7M), `nextvit_large` (57.9M), `resnet152c` (60.2M), `vit_betwixt_patch16_reg1_gap_256` (60.4M), `twins_pcpvt_large` (61.0M), `pvt_v2_b4` (62.6M), `resnetrs101` (63.6M), `resnet200d` (64.7M), `seresnet152d` (66.8M), `cait_s36_384` (68.4M), `wide_resnet50_2` (68.9M), `maxvit_small_tf_384` (69.0M), `swin_s3_base_224` (71.1M), `dm_nfnet_f0` (71.5M), `hgnet_base` (71.6M), `poolformer_m48` (73.5M), `pit_b_224` (73.8M), `coatnet_2_rw_224` (73.9M), `coatnet_rmlp_2_rw_224` (73.9M), `pit_b_distilled_224` (74.8M), `hrnet_w48` (77.5M), `tresnet_xl` (78.4M), `resnext101_64x4d` (83.5M), `repvgg_b3g4` (83.8M), `xcit_medium_24_p16_384` (84.4M), `xcit_medium_24_p16_224` (84.4M), `vit_base_patch16_dinov3` (85.6M), `deit_base_patch16_224` (86.6M), `vit_base_patch14_reg4_dinov2` (86.6M), `inception_next_base` (86.7M), `rdnet_base` (87.5M), `swin_base_patch4_window7_224` (87.8M), `davit_base` (88.0M), `focalnet_base_srf` (88.1M), `vit_base_patch32_224` (88.2M), `resnext101_32x8d` (88.8M), `gcvit_base` (90.3M), `fasternet_l` (93.5M), `seresnext101_32x8d` (93.6M), `seresnext101d_32x8d` (93.6M), `caformer_b36` (98.8M), and `convformer_b36` (99.9M).

E.2 Accuracy-versus-Complexity Experiment

The results reported in Fig. 4 are based on MLPs trained on Fashion-MNIST (Xiao et al., 2017). Four-layered MLPs of varying capacities were configured by doubling the number of hidden units. The base model had $[1024, 512, 256]$ hidden units; the three other versions were obtained by scaling the hidden units by the factor $s = [0.5, 2.0, 4.0]$, resulting in the four MLPs reported. All the models were trained for 100 epochs using 40k training samples, tested on 10k test samples using the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 3×10^{-4} and weight decay of 0.01 with a batch size of 128. All experiments were repeated over three random seeds, and we report the mean and standard deviation of all the metrics. Each experimental run to generate the data in Fig. 4 took about 4.5 hours on the hardware described, and through the project lifetime these experiments were conducted no less than 10 times, implying an overall compute budget of about 50 hours.

E.3 Learning Curve and Grokking Experiment

Learning curves reported in Fig. 5 follow a similar setup as reported above. Tiny-ViT (Wu et al., 2022) on CIFAR-10 (Krizhevsky, 2009) was trained for 300 epochs. Each Tiny-ViT experiment took about 6 hours, and these experiments were repeated over three random seeds.

For the Grokking experiment in Power et al. (2021), we use a 2-layered MLP with $[512, 256]$ hidden units to model the modular addition task $z = (x+y) \pmod P$ with $P = 97$. The dataset, consisting of the full P^2 Cartesian product, was partitioned into a 50% training split and a 50% validation split. The model was trained for 11k epochs, and metrics, including QuBD complexity, were logged during training/validation. The grokking experiments took about 5 minutes on the reported hardware and were repeated over three random seeds.

E.4 Continued Simplification at LLM Scales

We further study whether the relation between training dynamics and QuBD complexity extends to frontier AI models, using the Pythia large language model (LLM) family, which was trained using The Pile dataset comprising about 300B tokens (Biderman et al., 2023). We use the training loss and model checkpoints provided by the authors to estimate the QuBD complexity along the training trajectory across four parameter scales: 70M, 160M, 410M, and 1B parameters, illustrated in Fig. 13. Across all scales, we observe that QuBD complexity of the MSB is nearly flat and decreases later when the loss plateaus, matching the grokking experiment in Fig. 5 (**right**).

Furthermore, we note that binarized BDM complexity fails to track the complexity reduction, whereas QuBD, even with only the MSB-plane better tracks the reduction, consistent with the analysis in Sec. 3.1 and our findings in Fig. 5 (**left** and **center**). This exemplifies, additionally, that QuBD complexity does not coincide with sign-binarization, as discussed in Sec. 4.

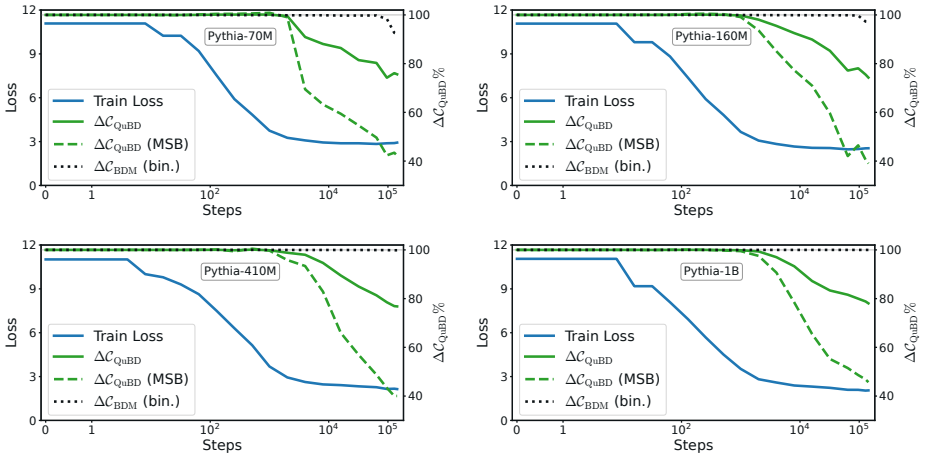


Figure 13: Learning curves along $\Delta\mathcal{C}_{\text{QuBD}}$, an MSB-plane only variant, and $\Delta\mathcal{C}_{\text{BDM}}$ evolution for increasing sizes of the Pythia model family with 70M, 160M, 410M, and 1B parameters.

E.5 Complexity per Layer Experiment

The results reported in Fig. 7 and Fig. 19 are based on pretrained ResNet-18 and ResNet-50 models sourced from the `timm` library (Wightman, 2019), compared against the same randomly initialized counterparts as in Appx E.1. For each layer, the weights are preprocessed as described in Appx. B.3 and quantized to 8 bits, clipping at the 99.9th percentile. The QuBD complexity ratio $\Delta\mathcal{C}_{\text{QuBD}}$ is then computed per layer rather than summed over the model, and is reported for the two most significant bit-planes (P7, P6). The experiment requires no dataset evaluation and took about 2 minutes to complete.

E.6 Post-Training Quantization Experiment

The results reported in Fig. 8 are based on five pretrained models sourced from the `timm` library (Wightman, 2019): ResNet-18, ResNet-50, ViT-B/16, EfficientNet-B0, and MobileNetV3-Large, all trained on ImageNet-1k (Deng et al., 2009). The full-precision (FP32) weights are quantized to 16 bits, clipping at the 99.9th percentile, and compared against the same randomly initialized counterparts as in Appx. E.1, following the weight tensor preprocessing described in Appx. B.3 QuBD complexity is reported for all 16 bit-planes. For post-training quantization (PTQ), the weights of all linear and convolutional layers are quantized per output channel, without any fine-tuning or calibration data. For 2 to 8 bits, we use a symmetric uniform quantizer with $2^b - 1$ levels whose scale is set by the channel's largest absolute weight. For 1 bit, each weight is replaced by its sign, scaled by the channel's mean absolute weight. Biases, normalization parameters, and activations are kept in FP32. All models were evaluated on the full ImageNet-1k validation set (50,000 samples). The FP32 baselines were evaluated the same way. The total compute time for this experiment was approximately 3.5 hours on the reported hardware.

F Additional Results

F.1 QuBD Complexity Ratio Across Eight Bit-Planes and Clipping Percentiles

Fig. 14 shows the QuBD complexity ratio across all eight bit-planes for 100 pretrained models from the `timm` library (Wightman (2019), see Appx. E.1), clipping at the 99.9th percentile as in the main text. Figs. 15 and 16 show the same models when clipping at the 99.99th percentile and without clipping. The complexity ratio generally decreases as clipping is relaxed, as extreme weights stretch the quantization range. For the most significant bit-planes, the median is 89% at the 99.9th percentile, 78% at the 99.99th percentile, and 35% without clipping.

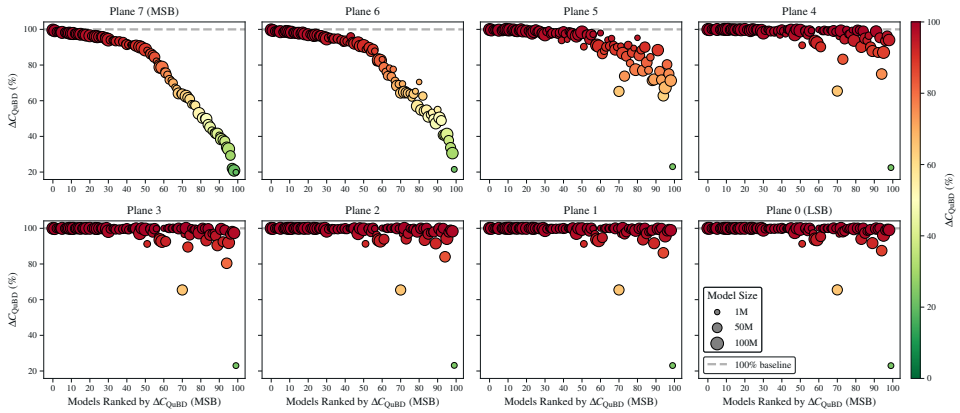


Figure 14: As Fig. 1 but shown for all eight bit-planes, clipping at the 99.9th percentile.

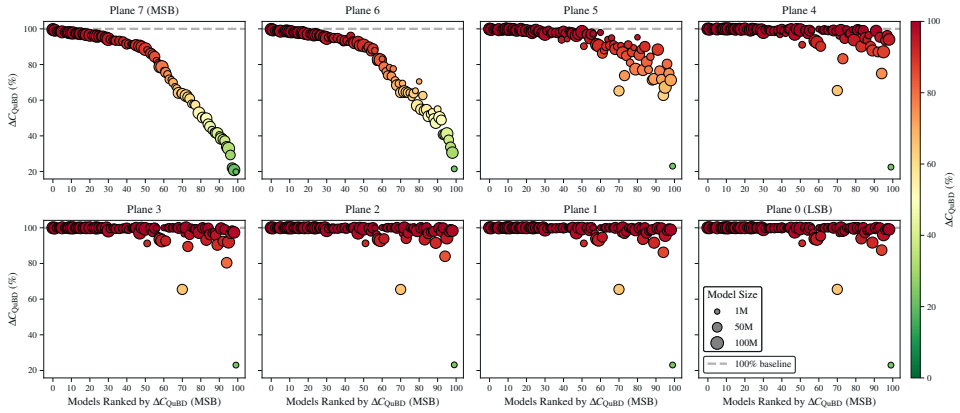


Figure 15: As Fig. 14, clipping at the 99.99th percentile. Models are ranked by their complexity ratio on the most significant bit-plane (Plane 7, MSB) for this setting.

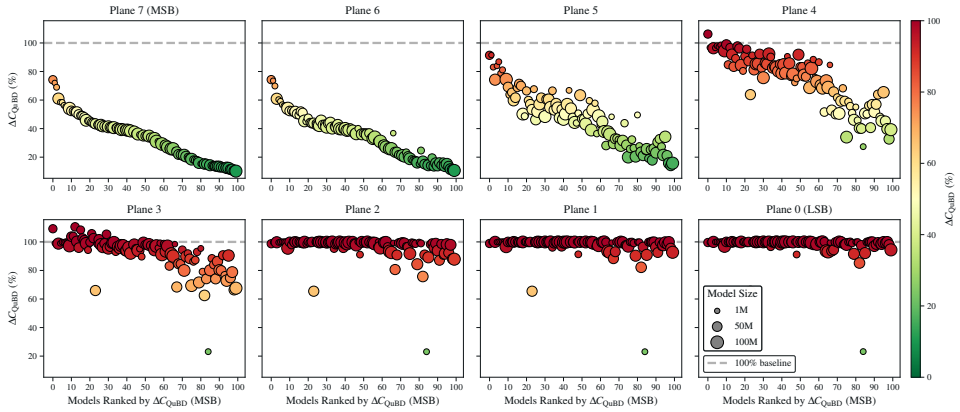


Figure 16: As Fig. 14, without clipping. Models are ranked by their complexity ratio on the most significant bit-plane (Plane 7, MSB) for this setting.

F.2 Robustness to Percentile-clipping on Complexity Reduction.

In relation to the quantized bit-plane decomposition, we note that the MSB would coincide when $w_{min} \approx w_{max}$, resulting in $w < 0 \rightarrow 0$ and $w > 0 \rightarrow 1$, similar to sign-binarization. One concern is then that the QuBD complexity is impacted by the weight distribution caused by percentile-clipping. For instance, a single outlier can shift the midpoint and flip the MSB for weights with smaller magnitude, giving rise to disparate structures. In Tab. 3 we compare to a symmetric range quantizer using 99.9th percentile-clipping at the final sample budget and see that the trends observed in Fig. 4 persist across all models. As both methods use similar CTM tables and reshaping, the remaining difference stems from the additional bit-planes retained by QuBD, which is consistent with the quantizer ablation in Sec. 4 (Tab. 1) and with the estimation-gap analysis in Sec. 3.1. Moreover, as meaningful complexity reduction happens across the five most significant planes, the larger contribution of the MSB in Figs. 1 and 8 does not suggest that the full QuBD or its MSB estimate reduces to a sign-binarized BDM.

Table 3: Complexity reduction with 99.9 percentile-clipping over three random seeds.

No. params.	$\Delta\mathcal{C}_{\mathrm{BDM}}$ (%)	$\Delta\mathcal{C}_{\mathrm{QuBD}}$ (%)
0.57M	40.84 ± 0.89	89.59 ± 0.48
1.46M	34.28 ± 0.67	94.88 ± 0.04
4.24M	25.90 ± 0.43	96.99 ± 0.10
13.71M	21.21 ± 0.25	96.28 ± 0.43

F.3 Model Size Effect on Complexity Reduction Ordering

Recall from Sec. 3.2 that the estimator fails to identify structure, and is uninformative, if the difference between $\mathcal{C}(\mathrm{x}_\rho)$ and $\mathcal{C}(\mathrm{x})$ goes towards zero. In the saturated regime, retaining only a few percent of the reference gap still yields a distinguishable measure of lower- and higher-complexity models. To identify, however, if there is an effect of the symbol size d exposed by the models in Fig. 1, we illustrate the $\Delta\mathcal{C}_{\mathrm{QuBD}}$ against parameter count in Fig. 17. In Tab. 4, we further compute the Spearman correlation of the relationship between model size and $\Delta\mathcal{C}_{\mathrm{QuBD}}$ across the models. We find a weak relationship ($|\rho| < 0.2$) across all bit-planes $\ell = 0, \ldots, 7$, which suggests that informativeness is primarily attributed to the retained bit-planes, consistent with Prop. 3.1 validated in Appx. C.4.

Table 4: Spearman correlation between parameter count and $\Delta\mathcal{C}_{\mathrm{QuBD}}(\%)$ across the 100 models in Fig. 1. We observe a weak relationship across all bit-lanes, suggesting the ordering of models with respect to complexity reduction does not stem from model size.

Plane	7 (MSB)	6	5	4	3	2	1	0 (LSB)
ρ	0.188	0.167	0.105	0.061	0.003	−0.042	−0.081	0.039

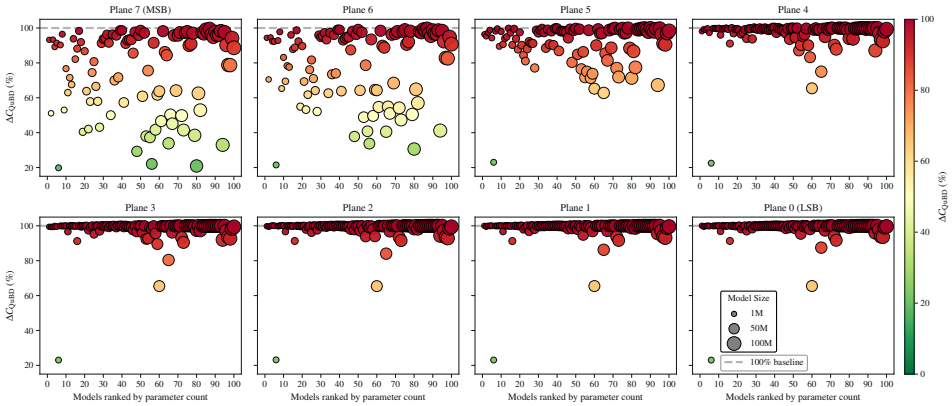


Figure 17: As Fig. 1. Models are ranked by their parameter count for this setting.

F.4 Quantized Bit-Planes with GZIP

As a sanity check, we replicate the results from Figs. 1, 4, and 5, using GZIP. GZIP estimates, when aggregated over bit-planes, however, are more discernible. Nevertheless, as estimates are not related to KCS complexity in any formal manner, unlike our QuBD complexity, we merely include them here for completeness.

F.5 QuBD Complexity Ratio for ResNet-50

Fig. 19 shows the normalized QuBD complexity per layer for ResNet-50 for the two MSB-planes (P7, P6). It shows significant differences in reduction across different layers, spanning close to $\Delta\mathcal{C}_{\mathrm{QuBD}} = 0\%$ and up to $\Delta\mathcal{C}_{\mathrm{QuBD}} = 100\%$. Experimental details can be found in Appx. E.5.

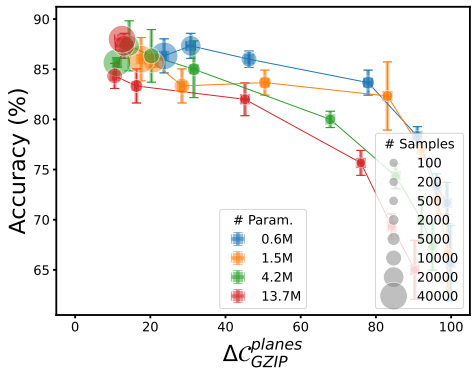


Figure 18: Relative GZIP complexity measured by summing over 8-bit planes, using the same parameters as for QuBD.

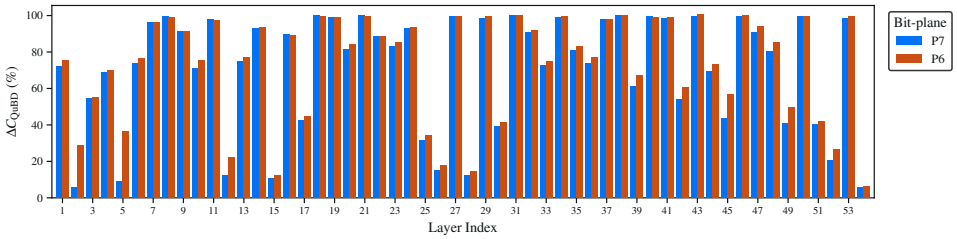


Figure 19: QuBD complexity ratio ΔC_{QuBD} per layer for ResNet-50 for the two most significant bit-planes (P7, P6).

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The introduction clearly states the contributions and the scope of the paper. All claims are directly supported by the theoretical results in Sec. 3 and/or the experiments in Sec. 3 and/or Sec. 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of the proposed QuBD method and the experiments are discussed in Sec. 4.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper provides complete and correct proofs with the full set of assumptions in Appx. C.2 (Prop. 3.1), Appx. C.3 (Prop. 3.2), Appx. D.2 (Prop. 3.3) and additional details for the latter is provided in Appx. D.1.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The code used for the experiments is provided via a GitHub repository with instructions for reproducing the experiments. Experimental details are also provided in Appx. E.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is made available via a GitHub repository with instructions for reproducing the experiments. All datasets used (Fashion-MNIST, CIFAR-10, ImageNet) are publicly available and can be downloaded without restrictions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All experimental settings necessary to understand the results are described in the main text and in Appx. E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports error bars and confidence intervals when suitable including in Tab. 1, Fig. 3 (right), Fig. 4, Fig. 5, Fig. 9, Fig. 10, and Fig. 11 (right), Fig 12 (right).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: This information can be found in Appx. E

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work conforms with the NeurIPS Code of Ethics. It involves no human subjects, raises no data-related concerns, and we foresee no negative societal impact.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: This work contributes to a better understanding of learning dynamics in deep neural networks, with direct positive societal impact through more principled model compression. By identifying redundant model capacity through algorithmic complexity analysis, QuBD can guide the development of smaller, more energy-efficient models, reducing the computational and environmental costs of deploying deep learning systems. We do not foresee direct negative societal impacts from this work, as it is foundational research focused on analysis tools rather than the development of new models or applications.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A].

Justification: The paper releases only analysis code and does not introduce new models, training data, or scraped datasets. We foresee no risk of misuse associated with this work.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All datasets, models, and code libraries used in this work are properly cited. We use publicly available datasets (Fashion-MNIST, CIFAR-10, ImageNet) and pretrained models from the `timm` library, all of which are cited accordingly. License information is not explicitly stated in the paper but all assets are used in accordance with their respective terms of use for academic research purposes.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper releases analysis code via a GitHub repository. The repository includes instructions for reproducing the main experiments.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A].

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: LLMs were not used as part of the core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.