Published at ProbML 2026

Probabilistic Inference for Buoyant Model Weights

Aleksanteri Sladek[1,2,†] **Martin Trapp**[3] **Arno Solin**[1,2]

[1]Department of Computer Science, Aalto University
[2]ELLIS Institute Finland
[3]KTH Royal Institute of Technology
[†]Correspondence to aleksanteri.sladek@aalto.fi

Abstract

Floating-point bit-string representations of parameters are *de facto* building blocks of neural networks, and their quantization into lower precision formats is increasingly used to reduce their computational requirements. The information loss that quantization induces in the model is, however, not well studied, and its detection relies on heuristics. We propose an information-theoretic approach for measuring quantization-induced information loss. By modelling the probability distribution of the underlying floating-point bit-strings that parameters consist of, we can measure information loss as the change in differential entropy across quantization levels. We empirically demonstrate that our method accurately captures quantization-induced information loss in neural networks trained on small benchmark regression data sets by showing a strong correlation between our information loss metric and the observed performance degradation across quantization levels.

1. Introduction

As modern machine learning models continue to grow in parameter count, reducing memory and compute through *quantization* has become increasingly common. Lowering the number of bits used to represent model parameters makes training and deployment substantially cheaper, and low-precision formats are now used throughout modern large-scale machine learning (Micikevicius et al., 2022; Dettmers et al., 2022; Frantar et al., 2022; Dettmers et al., 2023). Quantization, however, is inherently lossy. Most existing work studies its effect through downstream performance after quantization, which is useful but indirect. Arguably, this only serves as a proxy for the information loss that occurs in model parameters from quantization, and a more theoretically well-grounded approach would be beneficial for studying this effect more rigorously.

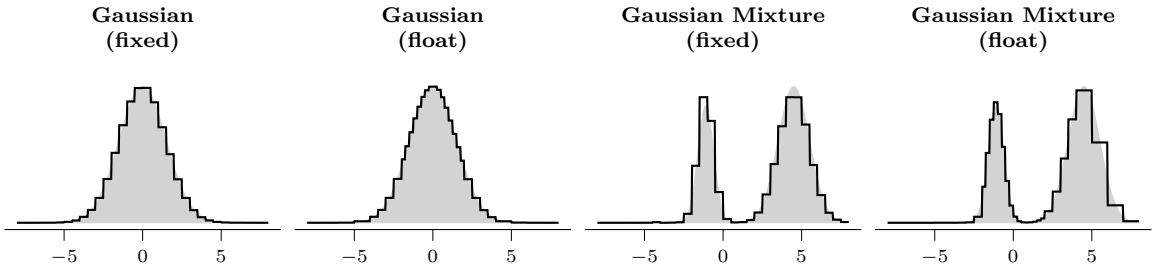

Figure 1: **BitVI on fixed-point and floating-point numbers.** 5-bit approximations of two single-parameter target densities (exact grey, approximation black line). The floating-point representation captures more detail around zero and less detail further away from zero.

This gap becomes even clearer when recalling that model parameters are never manipulated as ideal real numbers on a computer. They are always stored and processed as finite-length bit-strings (bit arrays) under a chosen number system, most commonly floating-point arithmetic (IEEE, 2019; Knuth, 1997; Sterbenz, 1974). A recently published method, BitVI (Sladek et al., 2025), builds directly on this observation and proposes modelling the posterior distribution of parameters over their underlying bit-string representations rather than over idealized continuous values. The motivation is simple: if learning and inference are ultimately carried out on discrete representations anyway, why not perform approximate inference in that same space?

In this work, we take this view to quantized model weights (see Fig. 1). We propose an information-theoretic measure of quantization-induced information loss based on entropy (Shannon, 1948). By leveraging the tractability and hierarchical structure of the bit-string posterior approximation provided by BitVI, we compute the differential entropy of the induced parameter posterior at different bit-widths and quantify information loss as the change in this entropy across quantization levels. In addition, we extend BitVI to floating-point bit-string representations (IEEE, 2019), which are the standard representation in modern machine learning systems, and consider neural networks in regression tasks, which were not studied in the original work.

2. Method

Floating-point representations Let $\theta \in \mathbb{R}$ represent a continuous parameter of a neural network f_θ, and let $\mathbf{b} \in \mathbb{B}^B$ be the B-bit *floating-point bit-string* $\mathbf{b} = \{b_{B-1}, b_{B-2}, \ldots, b_0\}, b_i \in \{0, 1\}$ that represents θ. An approximation of θ from $\mathbf{b}$ is given by the mapping function $f_{\text{float}} : \mathbb{B} \to \mathbb{R}$:

$$\theta = f_{\text{float}}(\mathbf{b}) = \underbrace{(-1)^{b_{B-1}}}_{\text{sign}} \cdot \underbrace{(2^{(\sum_{i=0}^{E} b_{i+1} 2^i - \kappa)})}_{\text{exponent}} \cdot \underbrace{(1 + 2^{\sum_{j=1}^{M} b_{(E+j)} 2^{-j}})}_{\text{mantissa}}, \tag{1}$$

where E and M represent the number of bits allocated for the exponent and mantissa, respectively, and κ is a bias term. This mapping function is the standard IEEE 754 floating-point representation (excluding special tokens) (IEEE, 2019), which is widely used in machine learning frameworks to represent model parameters.

Bit-strings as partitioning While Sladek et al. (2025) focused on *fixed-point* bit-string representations due to their simpler structure, we observe that the method can be extended to the more widely used floating-point bit-string representation. Similar to fixed-point bit-strings, floating-point bit-strings can also be viewed as a hierarchical binary partitioning of parameter space $\mathbb{R}$ into non-overlapping sub-intervals. The key difference is that instead of partitioning each sub-interval into equal-width sub-intervals via the arithmetic mean of the interval endpoints, the floating-point representation partitions space into unequal-width sub-intervals via a geometric mean of the endpoints. More specifically, exponential bits denote a geometric partitioning, whilst mantissa bits denote an arithmetic partitioning. See Alg. 1 and Fig. 3 for more details.

Distributions of Floats Let $C_\phi(\cdot)$ be a probability distribution over a length $\mathbf{b}$ floating-point bit-string which encodes parameter θ. We construct $C_\phi(\cdot)$ as a *deterministic* probabilistic circuit (PC) (Choi et al., 2020), which consists of a binary tree structure of computational nodes. The non-leaf nodes represent weighted summations with a set of weights $w_0, w_1 \in \phi$, and leaf nodes represent uniform density functions $\mathcal{U}(a_{\mathbf{b}_i}, b_{\mathbf{b}_{i+1}})$. The binary partitioning tree structure of the floating-point bit-string representation informs this structure, with endpoints $a_{\mathbf{b}_i}, b_{\mathbf{b}_{i+1}}$ of each of the 2^B input nodes corresponding to the intervals that two neighboring[1] floating-point bit-strings $\mathbf{b}_i, \mathbf{b}_{i+1}$ define. Furthermore, the weights ϕ can be interpreted as corresponding to the conditional probabilities $p(b_i = 1 \mid b_{i+1}, \ldots, b_{B-1})$ of a bit given its preceding bits. A key advantage of using this formulation

1. By a "neighboring" bit-string, we mean the bit-string which encodes the next nearest parameter value that can be represented by $\mathbf{b}$

Table 1: Average testing set RMSE scores (± s.d.) from a 5-fold CV experiment on UCI benchmark datasets. Methods are bolded based on a paired t-test ($p = 5\%$) with the best performing method, and show that floating-point BitVI performs similarly or better than fixed-point BitVI.

Dataset	(n, d)	10-bit BITVI (fixed)	10-bit BITVI (float)	6-bit BITVI (fixed)	6-bit BITVI (float)
YACHT	(308, 6)	**1.5752±0.5322**	**1.7018±0.6175**	2.0008 ± 0.5833	1.9141 ± 0.6273
KIN8NM	(8192, 8)	0.0964 ± 0.0056	**0.0781±0.0022**	0.1177 ± 0.0037	0.1190 ± 0.0026
BOSTON	(506, 13)	**3.4388±0.4204**	**3.4529±0.3189**	**3.4500±0.2917**	3.7481 ± 0.4802
ENERGY	(768, 8)	**0.7726±0.0376**	0.9750 ± 0.0575	1.3458 ± 0.0858	1.5155 ± 0.1149
PROTEIN	(45730, 9)	4.7017 ± 0.0442	**4.4250±0.0395**	4.8528 ± 0.0394	4.9367 ± 0.0308
POWER	(9568, 4)	4.1349 ± 0.0883	**4.0783±0.0889**	4.3936 ± 0.0860	4.3988 ± 0.1080
CONCRETE	(1030, 8)	**6.0720±0.3952**	**6.0300±0.3611**	**6.6486±0.5015**	6.7065 ± 0.4417
WINERED	(1599, 11)	**0.6409±0.0203**	**0.6321±0.0404**	**0.6322±0.0371**	**0.6364±0.0346**
NAVAL	(11934, 14)	0.0054 ± 0.0002	**0.0018±0.0002**	0.0095 ± 0.0003	0.0096 ± 0.0003
WINEWHITE	(4898, 11)	**0.7097±0.0139**	**0.7049±0.0106**	**0.7132±0.0127**	**0.7255±0.0126**

is that deterministic PCs allow for the computation of entropy exactly and efficiently (Vergari et al., 2021), which is a key aspect that our method leverages.

Variational approximation of bit-strings Given a data point $\mathbf{x} \in \mathcal{X}$, we aim to find the posterior distribution $p(\theta \mid \mathcal{X})$ of the neural network model $f_\theta(\mathbf{x})$ parameters θ. To this end, we use *variational inference* (Blei et al., 2017) to approximate this posterior with $C_\phi(\theta)$ as the variational distribution $q_\phi(\theta)$. Specifically, we optimize the parameters ϕ by maximizing the evidence lower bound (ELBO). We decompose the ELBO into an expected log-likelihood and an entropy term,

$$\mathcal{L}(\phi) = \underbrace{\mathbb{E}_{q_\phi(\theta)}[\log p(\mathcal{X}, \theta)]}_{\text{Expected log-joint}} + \underbrace{\mathbb{H}[q_\phi(\theta)]}_{\text{Entropy}} \approx \sum_{i=1}^{K} \log p(\mathcal{X}_i, \theta_i) + \mathbb{H}[q_\phi(\theta)], \qquad \theta_i \sim q_\phi(\theta). \tag{2}$$

Since the expected log-likelihood is intractable, we use Monte Carlo sampling to estimate it.

A measure of quantization-induced uncertainty With an approximation of the parameter posterior distribution over bit-strings in hand, we can then compute our proposed metric for quantization-induced information loss between bit-string lengths B and B' where $B' < B$. This computation is defined as:

$$\underset{B \to B'}{\Delta I} = \mathbb{H}[q_\phi^B(\theta)] - \mathbb{H}[q_\phi^{B'}(\theta)], \tag{3}$$

where the hierarchical structure and tractability of the variational distribution $q_\phi = C_\phi(\theta)$ allows for computing the differential entropy of the parameter posterior at the different bit-widths B and B'. As such, the difference in these two quantities directly measures the information lost when going from a B-bit representation to a B'-bit representation. Intuitively, a larger change in entropy indicates a larger information loss induced by the quantization operation.

3. Experiments

The proposed method was empirically validated with two experiments conducted on 10 benchmark regression data sets from the UCI machine learning repository (Kelly et al., 2025) via the *Bayesian Benchmarks*[2] testing framework. These regression tasks consist of predicting a 1D continuous value for data sets of various sizes and dimensionalities (See Table 1). We trained a small NN model consisting of two 16-unit hidden layers, with layer normalization (Ba et al., 2016) and the tanh(·) activation function applied after the hidden layers. For our posterior approximation, we used a Gaussian likelihood function $p(\mathbf{x} \mid \theta) = \mathcal{N}(f_\theta(\mathbf{x}), \sigma^2)$ with fixed variance σ^2, a Gaussian prior $p(\theta) = \mathcal{N}(\mu_p, \sigma_p^2)$, with μ_p and σ_p^2 selected such that the prior corresponds to the Kaiming initialization (He et al., 2015) and assumed that the NN parameters θ are independent of one another (mean-field assumption) in our variational family $q_\phi(\theta)$. We used 5-fold cross-validation, with a further 10% each training set reserved as a validation set used for early stopping, and report the average root mean squared error (RMSE) on the testing set across folds for each method and dataset.

2. Bayesian Benchmarks GitHub repository by Hugh Salimbeni et al.

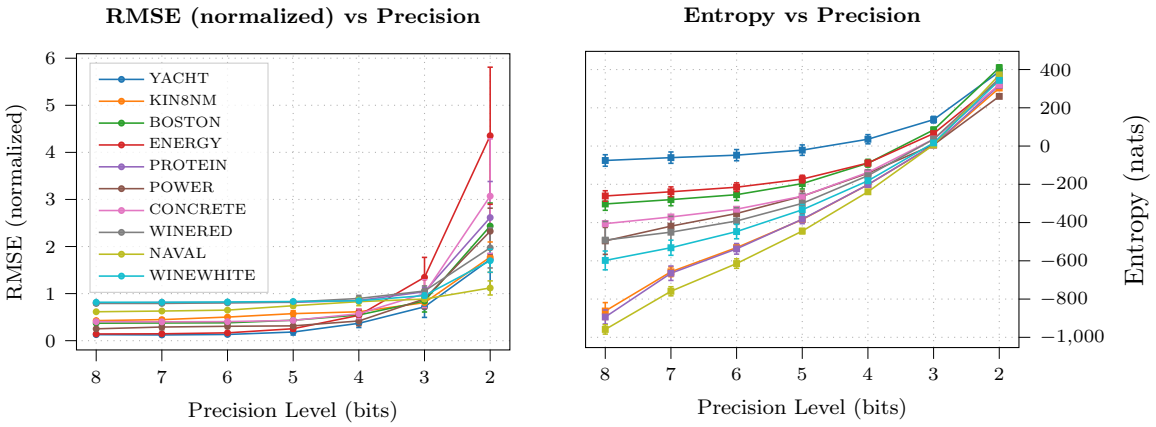

Figure 2: Comparison of the evolution of the average test set RMSE and entropy (error bars show s.d.) as a function of parameter precision level. With decreasing precision level, both RMSE and entropy increase at similar rates suggesting a correlation between the two.

Comparing fixed and floating-point BitVI We first compare the performance of the floating-point variant of BitVI with its original fixed-point variant. We trained 10-bit and 6-bit BitVI models with both the fixed-point and floating-point representations, and show the results in Table 1. The results on the 10-bit setting show that BitVI works similarly or better with floating-point representations. Reducing precision to 6-bits predictably leads to some performance degradation in both methods, but maintains surprisingly good performance given the quantization level.

Approximating quantization-induced information loss We evaluated whether our proposed metric for quantization-induced information loss accurately reflects the performance degradation observed with quantization. The results of this experiment are shown in Fig. 2, which shows the change in RMSE with increasing quantization (decreasing bit-width) from 8-bits to 2-bits, alongside the change in entropy. The metrics exhibit similar trends, with both RMSE and entropy remaining relatively stable until around 4-5 bits, after which they both increase rapidly.

4. Discussion & Conclusion

We proposed an approach based on the information-theoretic concept of entropy to measure the information loss that parameter quantization induces in machine learning models. By approximating the posterior distribution over the underlying bit-string representations of parameters with a hierarchical and tractable probabilistic model, we could directly compute how the entropy of our approximation changes at different quantization bit-widths. We also provided promising preliminary results suggesting that the proposed metric captures information loss that quantization induces.

Our perspective is also closely related to ideas from probabilistic numerics, where numerical computation itself is treated as a source of uncertainty and numerical tasks are cast as inference problems (Hennig et al., 2015; Cockayne et al., 2019; Hennig et al., 2022). While probabilistic numerics has mostly focused on problems such as integration and differential equations, the underlying message is directly relevant here. Finite-precision representation is not just a hardware detail, but rather a numerical constraint that shapes the uncertainty we can express about model parameters. Future work will explore scaling our method to study quantized large language models (Ma et al., 2024), investigate specialized number representations for deep learning, such as BFloats (Wang and Kanwar, 2019) and FP8 (Micikevicius et al., 2022), and study how to perform more targeted *mixed-precision* quantization using our proposed metric to identify the safest parameters to quantize. In particular, we draw inspiration from Van Baalen et al. (2020) who also propose a Bayesian approach for quantization, and elegantly unify quantization and pruning.

Acknowledgements

We acknowledge funding from the Research Council of Finland (362408 and 339730). This work was a part of Finland's Ministry of Education and Culture's Doctoral Education Pilot under Decision No. VN/3137/2024-OKM-6 (The Finnish Doctoral Program Network in Artificial Intelligence, AI-DOC). We acknowledge the computational resources provided by the Aalto Science-IT project. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, abs/1607.06450, 2016.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, University of California, Los Angeles (UCLA), 2020.

Jon Cockayne, Chris J Oates, Timothy John Sullivan, and Mark Girolami. Bayesian probabilistic numerical methods. *SIAM Review*, 61(4):756–789, 2019.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. `LLM.int8()`: 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, pages 30318–30332. Curran Associates, Inc., 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLORA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, pages 10088–10115. Curran Associates, Inc., 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, page 1026–1034. IEEE Computer Society, 2015.

Philipp Hennig, Michael A Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2179), 2015.

Philipp Hennig, Michael A Osborne, and Hans P Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press, 2022.

IEEE. IEEE standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The UCI machine learning repository. https://archive.ics.uci.edu, 2025.

Donald E Knuth. *The Art of Computer Programming: Fundamental Algorithms, Volume 1*. Addison-Wesley Professional, 1997.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

Aleksanteri Sladek, Martin Trapp, and Arno Solin. Approximate Bayesian inference via bitstring representations. In *Proceedings of the 41st Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 286 of *Proceedings of Machine Learning Research*, pages 3939–3957. PMLR, 2025.

Pat H. Sterbenz. *Floating-point Computation*. Prentice-Hall, 1974.

Mart Van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, pages 5741–5752. Curran Associates, Inc., 2020.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pages 13189–13201. Curran Associates, Inc., 2021.

Shibo Wang and Pankaj Kanwar. Bfloat16: The secret to high performance on cloud tpus. *Google Cloud Blog*, 4(1), 2019.

A. Appendix A: Additional Theoretical Details

In this section, we provide additional material for building intuition on how the floating-point bit-string representations work. In Alg. 1, we show an algorithm which iteratively partitions the parameter space $\mathbb{R}$. In Fig. 3, we show a visual comparison of how partitioning parameter space $\mathbb{R}$ with a fixed-point vs. a floating-point representation differs.

Algorithm 1: Floating-Point Partitioning Algorithm

Input: Bit-string $B = \{b_{B-1}, \ldots, b_0\}$, Exponent bits E, Mantissa bits M, No sign bit S
Output: Interval $[L, R]$

```
// Step 1: Constants and Bias Initialization
Bias ← 2^(E−1) − 1
L ← 2^(−Bias)                                   // Minimum non-zero magnitude
R ← 2^((2^E−1)−Bias+1)                          // Maximum magnitude
// Step 2: Exponent Partitioning (Geometric)
for i ← B − 1 to B − E − 1 do
    G ← √(L · R)                                // Partition using Geometric Mean
    if b_i == 1 then
        L ← G
    else
        R ← G
    end
end
// Step 3: Mantissa Partitioning (Arithmetic)
for i ← B − E − 1 to 0 do
    A ← (L+R)/2                                 // Partition using Arithmetic Mean
    if b_i == 1 then
        L ← A
    else
        R ← A
    end
end
return [L, R]
```

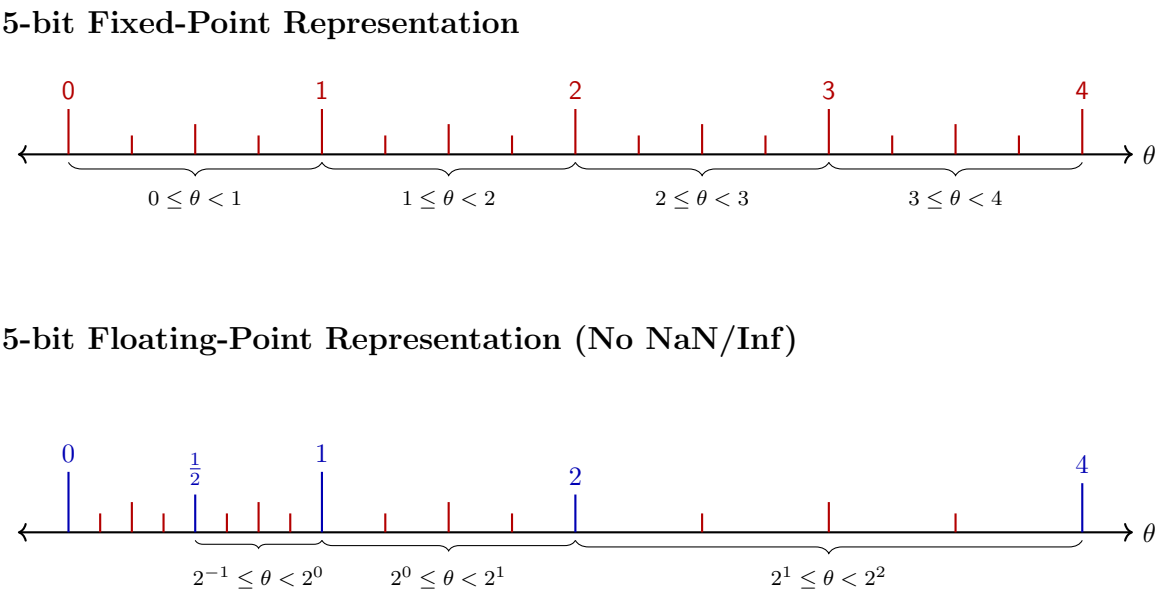

Figure 3: Comparison of how space is iteratively partitioned by 5-bit fixed-point vs. floating-point bit-string representations. For fixed-point bit-strings, the number line is partitioned into fixed-width intervals via an **arithmetic partitioning** function. For floating-point, exponent bits divide the number line via a **geometric partitioning** function, whereas mantissa bits perform arithmetic partitioning. In this example, the 5-bit fixed-point bit-string illustrated has 2 integer bits and 2 fractional bits. The floating-point bit-string has 2 exponent bits, 2 mantissa bits, a bias of 2, and skips reserving an exponent for NaN/Infinity to maximize the representable range up to the 4.0 bound.