Schrödinger's FP: Training Neural Networks with Dynamic Floating-Point Containers

Miloš Nikolić [1 2] **Enrique Torres Sanchez** [1] **Jiahui Wang** [3 *] **Ali Hadi Zadeh** [4 *] **Mostafa Mahmoud** [5 *]
Ameer Abdelhadi [6 *] **Kareem Ibrahim** [1] **Andreas Moshovos** [1 2]

Abstract

The transfer of tensors from/to memory during neural network *training* dominates time and energy. To improve energy efficiency and performance, research has been exploring ways to use narrower data representations. So far, these attempts relied on user-directed trial-and-error to achieve convergence. We present methods that relieve users from this responsibility. Our methods dynamically adjust the size and format of the floating-point containers used for activations and weights during training, achieving adaptivity across three dimensions: i) which datatype to use, ii) on which tensor, and iii) how it changes over time. The different meanings and distributions of exponent and mantissas lead us to tailored approaches for each. We present two lossy *pairs* of methods to eliminate as many mantissa and exponent bits as possible without affecting accuracy. *Quantum Mantissa* and *Quantum Exponent* are machine learning compression methods that tap into the gradient descent algorithm to *learn* the minimal mantissa and exponent bitlengths on a per-layer granularity. They automatically *learn* that many tensors can use just 1 or 2 mantissa bits and 3 or 4 exponent bits. Overall, the two machine learning methods reduce the footprint by $4.74\times$. Alternatively, *BitWave* observes changes in the loss function during training to adjust mantissa and exponent bitlengths network-wide, yielding a $3.19\times$ reduction in footprint. Finally, we present an optional method, *Gecko*, to exploit the naturally emerging, lop-sided exponent distribution to *losslessly* compress resulting exponents from *Quantum Exponent* or *BitWave* and, on average, improve compression rates to $5.64\times$ and $4.56\times$.

1 Introduction

While training neural networks is both computationally and data demanding, it is the memory transfers to off-chip DRAM for *stashing* (i.e., saving and much later recovering) activation and weight tensors that dominate execution time and energy (Jain et al., 2018). The per batch data volume easily surpasses on-chip memory capacities, necessitating off-chip DRAM accesses which are up to two orders of magnitude slower and more energy expensive (Horowitz, 2014). Reducing this overhead has been receiving attention throughout the software/hardware stack and is also our goal.

The most direct way to reduce tensor volume is by using datatypes that use fewer bits per value, e.g., BFloat16 (Kalamkar et al., 2019), half-precision floating-point (FP16), dynamic floating-point (Das et al., 2018), flexpoint (Köster et al., 2017)), or fixed-point (Das et al., 2018; Micikevicius et al., 2018; NVIDIA; Drumond et al., 2018)). This reduces memory traffic and footprint, improving energy efficiency and execution time. In the past, training typically used single precision 32b floating-point (FP32), as it was believed to yield the best accuracy. However, recent research has shown that using more compact datatypes can still achieve good results while reducing memory usage. Some works have even pushed the limits of datatype efficiency by using 8b (Wang et al., 2018b) and 4b (Sun et al., 2020) datatypes in *certain* cases. Industry is even exploring the use of 8b floating point with different mantissa/exponent ratios to meet the specific needs of tensors (Micikevicius et al., 2022) and even shorter formats (Rouhani et al., 2023b). As industry is expanding support for leaner datatypes the following challenges remain:

- To achieve convergence current approaches rely *exclusively* on trial-and-error: It is up to the user to carefully select which datatype to use for each tensor. This often necessitates changes to the training recipe and the inclusion of additional operations such as loss scaling (NVIDIA). Convergence is not guaranteed and can be evaluated only *post mortem*.

[*] Work completed at the University of Toronto [1] Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada [2] Vector Institute for Artificial Intelligence, Toronto, Canada [3] Qualcomm, Toronto, Canada [4] 1QBit, Toronto, Canada [5] AMD, Toronto, Canada [6] Department of Computer Engineering, McMaster University, Hamilton, Canada. Correspondence to: Miloš Nikolić <milos.nikolic@mail.utoronto.ca>, Enrique Torres Sanchez <enrique.torres@mail.utoronto.ca>.

Proceedings of the 7^{th} MLSys Conference, Santa Clara, CA, USA, 2024. Copyright 2024 by the author(s).

- Universally, all methods store weights in full-precision as the backward pass performs minuscule updates that cannot be represented with the leaner datatype.
- The datatypes are statically chosen offering no opportunity to amend the choice if accuracy suffers (e.g., significant drop with deeper networks identified by IBM (Sun et al., 2020)).
- Even where successful, these methods still use a scant repertoire of bitlengths (e.g., tensors fitting in 5b have to use 8b, a nearly 2x increase), leaving a lot of opportunity for memory overhead reduction untapped.
- They require hardware changes to allow computation with the leaner datatypes.

This work automates and fuses *into training itself* the process of datatype discovery improving execution time and energy efficiency. Given that floating-point remains the datatype of choice to ensure convergence, we focus on *automatic* floating-point datatype selection with the goal being to reduce memory traffic during training. Our methods:

- *Dynamically* and *continuously* adjust the *mantissa* and the *exponent bitlengths* for floating-point activations and/or weights for stashed tensors, and do so *transparently* at no additional burden to the user.
- Are adaptable across three dimensions: The first two automate what is currently done by hand: *which* datatype to use for *which* tensor. Uniquely, our methods adapt these datatypes over *time*.
- Adapt the exponent bitlengths to their actual content using only as many bits as necessary to store their value. Most exponents end up using a lot fewer bits than statically selected datatypes.
- Store values in memory with only as many bits as necessary while expanding values to the closest available datatype supported by the accelerator.
- In addition to accelerating training, our methods can inform efforts for selecting more efficient datatypes for inference such as that by Micikevicius et al. (2022), Rouhani et al. (2023b) or Sun et al. (2020).
- As a by-product, quantize the networks to efficient datatypes which benefits inference.

Our solution is *Schrödinger's FP*, a *family* of two methods that learn exponents and mantissa bitlengths, and an *optional* lossless exponent compression method *Gecko*:

Quantum Mantissa & Exponent: The first method comprises *Quantum Mantissa* (*QM*) and *Quantum Exponent* (*QE*), and harnesses the training algorithm itself to *learn* on-the-fly the per tensor mantissa and exponent bitlengths which it continuously adapts per batch. *QM* and *QE* introduce a learning parameter per tensor and a regularizer that include the effects of the mantissa and exponent bitlengths. Learning the bitlengths incurs a negligible overhead compared to the resulting reduction in off-chip traffic. Experiments show that: 1) they reduce bitlengths considerably, more so for mantissas, 2) the bitlengths vary per tensor and 3) fluctuate throughout, capturing benefits that wouldn't be possible with a static network-wide choice of datatype.

***BitWave*:** *BitWave* approaches the training as a black-box observing the effect of adjusting mantissa and exponent bitlengths on its progress. It uses a simple linear regression of a history of losses (observed per-batch) to adjust the mantissa and exponent bitlengths for the whole network. As long as the network seems to be improving, *BitWave* will attempt to shorten them; otherwise, it will increase them. *BitWave* proves effective, albeit with lower bitlength reductions compared to *QM+QE*, since: 1) they harness the training process to learn the optimal bitlengths, and 2) they adjust bitlengths per layer whereas *BitWave* does so network-wide to reduce the search space.

***Gecko*:** On top of *QM+QE* and *BitWave* exponent bitlength reduction, we introduce an *optional* method, *Gecko*, which exploits their biased distribution that naturally occurs during training (Awad et al., 2021). *Gecko* stores exponents using only as many bits as necessary to represent their value, outperforming any statically chosen bitlength. *Gecko* chooses the bitlength per group of values to reduce metadata overhead achieving high encoding efficiency. Encoding values in DRAM using variable length containers is standard practice in systems for deep learning, (Han et al., 2016c; Lascorz et al., 2019; Han et al., 2016b).

Reducing Off-Chip Traffic: We demonstrate that our methods boost energy efficiency and performance by transparently encoding values as they are being stashed to off-chip DRAM, and decoding them to their original format as they are being read back. To do so, we introduce *(de)compressor* units in front of the memory controller leaving the rest of the on-chip memory hierarchy and compute cores unchanged. Future work can investigate using *Schrödinger's FP* to boost computation throughput as well.

To maximize benefits, we present a hardware-assisted implementation of *Schrödinger's FP* (a software-only implementation is possible as well but is left for future work); the inclusion of specialized hardware units is now commonplace among all hardware vendors. Appendix A presents efficient hardware (de)compressors that operate on groups of unmodified floating-point values. The units accept external mantissa and exponent length signals and pack values maintaining DRAM-friendly long, sequential accesses. The decompressors expand such compressed blocks back into the original floating-point format.

Schrödinger's FP will generally work in conjunction with methods that partition, distribute, or reschedule the training work to improve energy efficiency and performance, or that can improve accuracy for a preselected datatype. We highlight the following key contributions and experimental findings from *Schrödinger's FP*:

- We introduce two machine learning based methods: *Quantum Mantissa* (*QM*) and *Quantum Exponent* (*QE*), which harness the training algorithm itself to dynamically learn per-tensor mantissa and exponent bitlengths, adjusting them continuously with each batch. *QM+QE* reduces the memory footprint by $4.74\times$ on average (range: $3.35\times$ to $13.23\times$). The *QM+QE* experiments demonstrate variability in the mantissa and exponent bitlengths across different tensors, thereby highlighting the superiority of this per-tensor approach.
- We introduce two loss observation based methods: *BitWave Mantissa* (*BWM*) and *BitWave Exponent* (*BWE*), which approach the training process as a block-box, and observe the effect of adjusting mantissa and exponent bitlengths, via the loss function. *BitWave* reduces the memory footprint without noticeable loss of accuracy by $3.19\times$ on average (range: $2.24\times$ to $8.91\times$). Crucially, *BitWave* stays transparent to the process and has negligible overhead.
- We introduce *Gecko*, a lossless exponent group compression method for training. Our work shows that this method can further boost the *QM+QE* and *BitWave* footprint reduction to $5.64\times$ on average (range: $3.73\times$ to $17.66\times$) and $4.56\times$ on average (range: $3.07\times$ to $9.74\times$), respectively.
- Indicatively, for an accelerator using BFloat16 and with a peak throughput of 16 TFLOPS, $\mathrm{S}FP_{\mathrm{Q}+G}$ and $\mathrm{S}FP_{\mathrm{B}W+G}$ improve energy efficiency by $3.07\times$ and $2.71\times$. When the accelerator uses FP8 instead, our aforementioned methods improve energy efficiency by $2.26\times$ and $2.00\times$ respectively.

2 Training with Efficient Datatypes

The question of which *training* datatype strikes the right balance among accuracy, energy and time remains open. Recently, we have seen success in training with more compact floating-point such as half-precision FP16 and BFloat16 (Kalamkar et al., 2019). These approaches can match single-precision (FP32) accuracy and provide significant cost reduction, however, they are still over-provisioned and leave potential unexploited. There has been *limited* success at using very small datatypes with 8b (Wang et al., 2018b) and 4b (Sun et al., 2020) extremes for *some* cases. Similarly, major hardware manufacturers are investigating how to use narrower floating point with different mantissa/exponent ratios according to perceived needs of tensors (Micikevicius et al., 2022; Rouhani et al., 2023b). These datatypes are often tailored to specific network architectures and current selection approaches cannot match FP32 accuracy outside of a narrow subset of shallow networks. Other energy efficient datatypes have been proposed including dynamic floating-point (Das et al., 2018), flexpoint (Köster et al., 2017), hybrid block floating-point (Drumond et al., 2018; Rouhani et al., 2023a) and combinations with other datatypes like fixed-point (Das et al., 2018; Micikevicius et al., 2018; NVIDIA).

These *tailored* methods require careful trial-and-error investigation of *where*, *when*, and *which* datatypes to use. This is challenging because different tensors, tasks, architectures, or layers *require* different datatypes. The methods require full trial-and-error training runs and *post mortem* analysis as whether the choice of datatypes is viable. Moreover, since the datatypes are statically chosen they offer no opportunity to amend the choice if accuracy suffers (e.g., significant drop with deeper networks (Sun et al., 2020)).

It's important to recognize that for preselected datatype methods, *two* key decisions have to be made: 1) the meaning of the bits, and 2) the required number of bits. The first decision — defining the meaning of the bits — has usually been the bigger contribution. It involves choosing between options like integer or exponential representations, shared or individual exponents for floating points, or using lookup tables. Traditionally, the number of bits is determined through experience or experimentation. Our methods automate this selection process for any chosen representation. Moreover, the principles from *Schrödinger's FP* are independent of the bit meaning and can determine the optimal number and type of bits required. This enhances the process with automation, adaptability, and granularity, moving beyond the conventional, fixed approach.

Adaptable methods are gathering attention. *Open-loop* methods modify the datatype based on a predetermined schedule but require trail-and-error runs to find an adequate schedule. *Closed-loop* solutions that monitor some metric other than loss or task accuracy (e.g., quantization error) comparing against a preset allowable error schedule (based on time, layer depth, or other network features) run into the same issue (Qian Zhang et al., 2022; Zhang et al., 2020).

ACGC (Evans & Aamodt, 2021) determines leaner datatypes to use in mixed-precision fixed-point quantization for *activations*. It periodically determines the maximum permissible quantization error bound for each *activation* tensor based on a user-selected maximum allowable increase in loss and adjusts the bitlength they use. ACGC can not compress weights and is not applicable where weights dominate such as most natural language processing networks. Determining the permissible bounds is also expensive, however, its overhead is kept down by performing it infrequently.

Obviously, knowing in advance which compact datatypes to use, and when, would be the best. However, given that this goal still eludes us, our work asks whether we can harness the training process itself to automatically *learn* them, 1) *automatically* tailoring datatypes to each tensor, layer, and network, and 2) continuously adjusting them as training progresses, adapting to the changing needs.

We present a fully automatic closed-loop solution that tracks the loss. Our approach redefines mantissa and exponent quantization to make them differentiable and includes the reduction of datatype size as part of the objective of gradient descent, without high overhead.

Effective closed-loop solutions for finding the most efficient datatype exist for *inference*. They use reinforcement learning (Wang et al., 2018a), differentiable datatype definitions (Nikolić et al., 2020; Huang et al., 2022), architecture search (Wu et al., 2018), learnable parameters for every weight bit (Yang et al., 2021), and profiling (Nikolić et al., 2018), etc., and have been proposed for fixed-point *inference* mixed precision quantization. However, all of these are too expensive for training and their overheads would overshadow the benefits of a more compact training datatype. Moreover, some are specifically targeting weights or activations, and can not adapt to different architectures where the main footprint contributors may change (weight vs activation heavy cases).

3 ADJUSTING VALUE CONTAINERS

In general, maintaining accuracy on most real-world tasks requires floating-point-based training. These formats comprise a sign S, a mantissa M, and an exponent E:

$$V(S,M,E) = (-1)^S \times (1+M) \times 2^E \quad (1)$$

Each part is differently distributed and requires unique approaches to effectively compress. The sign S only needs 1 bit and when V is limited to only positive numbers, it can be omitted. M, including its implied one, is the fractional part of the multiplier and, denormals aside, has a range $[1, 2)$. Reducing M's length reduces the *precision* of the full value. Finally, E is the exponent of the second multiplier. Reducing E's length narrows the *range* of the full value:

$$V(S,M,E) \in [-V_{max}, -V_{min}] \cup \{0\} \cup [V_{min}, V_{max}] \quad (2)$$

where V_{max} and V_{min} are the absolute values of the limits of V with the exponent range $[E_{min}, E_{max}]$ and maximum mantissa M_{max}:

$$V_{max} = (1+M_{max}) \times 2^{E_{max}} \quad (3) \qquad V_{min} = 2^{E_{min}} \quad (4)$$

Sections 3.1 and 3.2 present respectively machine learning and hardware-inspired approaches for learning mantissa and exponent bitlengths. Both methods omit the sign bit whenever possible. Section 3.3 complements either approach with an optional lossless exponent compression method.

We study *Schrödinger's FP* with ResNet18, ResNet50 (He et al., 2015) and MobileNet V2 (Sandler et al., 2018) trained on ImageNet (Russakovsky et al., 2014), DLRM (Naumov et al., 2019) trained on Kaggle Criteo, Vision Transformer (Dosovitskiy et al., 2020) pretrained on Cifar10 (Krizhevsky, 2009), BERT (Devlin et al., 2018) finetuned on GLUE (Wang et al., 2019) and GPT–2 (Radford et al., 2019) finetuned on Wikitext 2 (Merity et al., 2016). We report detailed results with ResNet18 and conclude with overall results for all models.

3.1 Machine Learning Approach

Quantum Mantissa and *Quantum Exponent* learn mantissa and exponent bitlengths, respectively. Both use inexpensive procedures for both the forward and the backward pass of training and rely on making quantization differentiable and penalizing the larger bitlengths in the loss function. We begin by defining a conventional quantization scheme for integer mantissa and exponent bitlengths in the forward pass, and then expand it to the non-integer domain to allow gradient descent to learn bitlengths. A parameterizable loss function guides this learning by penalizing larger bitlengths. We then touch on the compute overhead of our methods and the plan for the final selection of mantissa bitlengths. Ultimately, we demonstrate the benefits of this approach on memory footprint during ImageNet training.

***Quantum Mantissa* (*QM*):** The greatest challenge for learning bitlengths is that they represent discrete values with no obvious differentiation. To overcome this, we define our quantization for non-integer bitlengths, starting with an *integer quantization* of the mantissa M with n_m bits by removing all but the top n_m bits:

$$P(M, n_m) = M \wedge (2^{n_m} - 1) << (m - n_m) \quad (5)$$

where $P(M, n_m)$ is the mantissa with bitlength n_m, m the maximum bitlength and $\wedge$ a bitwise AND.

This scheme does not allow the learning of bitlengths with gradient descent due to its discontinuous and non-differentiable nature. To expand the definition to real-valued $n_m = \lfloor n_m \rfloor + \{n_m\}$, the values used in inference during training are *stochastically* selected between the nearest two integers with probabilities $\{n_m\}$ and $1 - \{n_m\}$:

$$P(M,n_m) = \begin{cases} P(M, \lfloor n_m \rfloor), & \text{w/ prob. } 1-\{n_m\} \\ P(M, \lfloor n_m \rfloor + 1), & \text{w/ prob. } \{n_m\} \end{cases} \quad (6)$$

where $\lfloor n_m \rfloor$ and $\{n_m\}$ are floor and fractional parts of n_m.

This mantissa approach faithfully represents the relationship between bitlength and precision in an *inexpensive* way. The overhead is limited to the single bitlength parameter and a random number (in the forward pass) per value group (e.g., a tensor), and a single multiply-accumulate operation (in the backward pass) per value.

***Quantum Exponent* (*QE*):** The exponent range is parameterized as follows:

$$R(V,V_{max},V_{min})=\begin{cases}-V_{max}, & V\in(-\infty,-V_{max})\\ V, & V\in[-V_{max},-V_{min}]\\ -V_{min}, & V\in(-V_{min},-V_{min}/2]\\ 0, & V\in(-V_{min}/2,V_{min}/2)\\ V_{min}, & V\in[V_{min}/2,V_{min})\\ V, & V\in[V_{min},V_{max}]\\ V_{max}, & V\in(V_{max},\infty)\end{cases} \tag{7}$$

where V_{max} and V_{min} are boundaries from Equation 2.

The partial derivatives of this function with respect to V, V_{max} and V_{min} are:

$$\frac{\partial R}{\partial V}=\begin{cases}0, & V\in(-\infty,-V_{max}]\\ 1, & V\in(-V_{max},V_{max})\\ 0, & V\in[V_{max},\infty)\end{cases} \tag{8}$$

$$\frac{\partial R}{\partial V_{max}}=\begin{cases}-1, & V\in(-\infty,-V_{max}]\\ 0, & V\in(-V_{max},V_{max})\\ 1, & V\in[V_{max},\infty)\end{cases} \tag{9}$$

$$\frac{\partial R}{\partial V_{min}}=\begin{cases}0, & V\in(-\infty,-V_{min}]\\ -1, & V\in(-V_{min},-V_{min}/2]\\ 1, & V\in(-V_{min}/2,0)\\ -1, & V\in[(0,V_{min}/2)\\ 1, & V\in[V_{min}/2,V_{min})\\ 0, & V_{min}<V\end{cases} \tag{10}$$

The next challenge of finding the exponent bitlength gradient is to connect the value range with the exponent range:

$$V_{max}=(1+M_{max})\times 2^{E_{max}} \quad (11) \qquad V_{min}=2^{E_{min}} \quad (12)$$

Where M_{max} is the largest possible mantissa, E_{max} is the largest possible exponent, and E_{min} is the smallest possible exponent. For simplicity, we choose our exponent range to be symmetrical around 0:

$$E_{min}=-2^{n_e^i-1} \quad (13) \qquad E_{max}=2^{n_e^i-1}-1 \quad (14)$$

where the integer n_e^i is the integer exponent bitlength. The bias can also be learned, however, this is not essential as the important exponents will be around 0. As with *QM*, we expand this definition to the continuous domain stochastically:

$$n_e^i=\begin{cases}\lfloor n_e\rfloor, & \text{w/ prob. } 1-\{n_e\}\\ \lfloor n_e\rfloor+1, & \text{w/ prob. } \{n_e\}\end{cases} \tag{15}$$

where n_e is the learnable exponent bitlength.

Similar to *QM*, this approach faithfully represents the relationship between exponent bitlength and the range in an *inexpensive* way. Its overhead is limited to the single bitlength parameter and a random number (in the forward pass) per value group sharing a datatype (e.g., tensor), and a single operation (in the backward pass) per value.

Finally, in order to obtain the fully quantized value we first bound the input with R to remove the exponent bits and then apply P to remove the mantissa bits.

Datatype Learning: These schemes are applied to each activation and weight tensor separately. Since the minimum bitlength is 0, n_m and n_e are clipped at 0. This extension of the bitlengths in the continuous domain allows the loss to be differentiable with respect to both E and M bitlengths.

The formulae above are applied during the forward pass. Quantized values are saved and used in the backward pass. This strategy reduces the footprint of training because only quantized values are used in forward and backward passes.

Loss Function: We augment the loss L to penalize M and E bitlengths by adding a weighted average of their volume:

$$L=L_l+\gamma_m\times\sum^{i}(\lambda_i\times n_m^i)+\gamma_e\times\sum^{i}(\lambda_i\times n_e^i) \tag{16}$$

where L_l is the original loss, γ_m and γ_e are regularization coefficients determining quantization aggressiveness, λ_i is the importance of the i^{th} group of values (one per tensor), and n_m^i and n_e^i are the mantissa and exponent bitlengths of the activations or weights in that tensor.

Competing Objectives: Our augmented loss adds a competing objective for training. Overemphasizing bitlength choice may sacrifice task performance, while underemphasizing it may sacrifice potential gains. Balancing the objectives via γ selection proves straightforward for two reasons. First, from our experience, selecting γ such that the bitlength loss component is 1-2 orders of magnitude smaller than the main task objective loss is enough to squeeze out most of the reduced datatype benefits whilst being sufficiently small to *not* adversely influence final accuracy. Second, finding the best γ isn't necessary since learning the bitlengths is a very coarse task, and at the end, the bitlengths have to be rounded to appropriate integer ones. For all experiments setting both γ_m and γ_e to 0.1 proved sufficient.

Target Criteria: Our loss function can target any quantifiable criteria by a suitable selection of λ_i's. Since our goal is to minimize the total footprint of training, we weigh each tensor according to its memory footprint.

Overhead: *QM* and *QE* add minimal computational and memory overheads. In the forward pass, random numbers are needed at a chosen granularity as per eq. 6 and 15. Our experiments show that a random number per tensor per batch is sufficient and is of a negligible cost.

To update the bitlength parameters in the backward pass, we need to compute their gradients. These are a function of the quantized values and gradients, which are calculated during the regular backward pass. The extra calculations are proportional to the number of values. This overhead is negligible in comparison to the total number of computations. For instance, for ResNet18 the overhead is less than 1%.

The only new parameters that are stashed are the four floats per layer (mantissa and exponent bitlength for weights and activations), negligible in comparison with the total footprint. All other values are consumed as they are produced.

Bitlength Selection Schedule: *QM+QE* use non-integer bitlengths. We arrive at integer bitlengths, by disabling learning bitlengths when they are not needed at which point we *round up* the bitlengths and freeze them. Our experiments show that bitlengths converge quickly to the final ones within a couple of epochs. Accordingly, we freeze the bitlengths after epoch 5. This avoids the small overhead for most of the training. In our experiments, we found that when both methods are used concurrently some tensors *might* need more bits later in training. Accordingly, we re-enable *QM* and *QE* for an additional 5 epochs on every learning rate change. This allows precision to increase where necessary to accommodate the reduction in update magnitudes. Regardless of whether *QM* and *QE* are enabled or disabled, the benefits of reduced bitlengths apply throughout training. This ”fancy” schedule is not fully needed. Experiments where we fixed the bitlengths after 5 epochs still converged and achieved *slightly* lower accuracy.

Evaluation: Bitlengths and Accuracy: We report measurements for per-layer weights and activations quantized separately using a loss function weighted to minimize overall memory footprint. We train ResNet18 on the ImageNet dataset over 90 epochs, with regularizer strength of 0.1, learning rate of 0.1, 0.01 and 0.001 respectively at epochs 0, 30, and 60 and weight decay of 0.0001.

Both *QM* and *QE* excel at minimizing the memory footprint whilst not introducing accuracy loss. Figure 1a shows that throughout training, our methods introduce minimal changes in validation accuracy converging to a solution within 0.4% of the FP32 baseline. Minor accuracy loss occurs when the methods are actively pushing bitlengths to their limits. Any loss is quickly regained when bitlenghts are frozen and rounded up since this relaxes the value range.

***QM*:** Figure 1b shows how *QM* quickly (within a couple of epochs) reduces mantissas below 2b on average. The large spread in bitlengths across layers shows that *QM*'s granular, per tensor approach is the right choice for boosting benefits. In comparison, FP8 would use 2-3b (out of 8) everywhere (Micikevicius et al., 2022). While *QM sometimes* allocates more than 3b for some tensors, this slack boosts overall footprint reduction since it enables shorter bitlengths for larger tensors. Finally, the results show a minor increase of bitlengths across period boundaries of our bitlength learning schedule. The total training cumulative mantissa footprint is reduced to 8.4% of the FP32 mantissa footprint (8.3% for activations and 9.8% for weights).

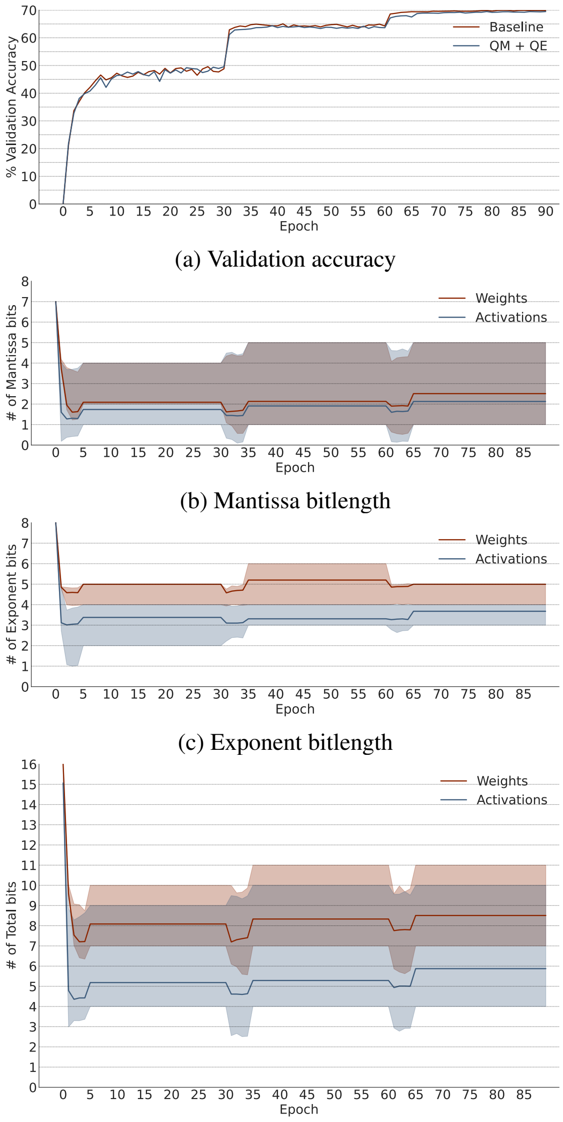
***QE*:** Figure 1c shows that learning exponent bitlengths via *QE* exhibits similar behavior. Bitlengths quickly converge to 4b or less for activations, and on average down to around 5b for weights. In comparison, FP8 would use 4-5b (out of 8) everywhere, a fair choice for network-wide bitlength (Micikevicius et al., 2022). *QE* sometimes uses longer exponents for some tensors enabling short exponents for large tensors. As a result, *QE* outperforms FP8 in exponent footprint. Compared to mantissas, the spread in exponent bitlengths across layers is lower yet significant while there is a more noticeable increase of bitlengths from one learning period to the next. The cumulative training memory footprint is reduced to 43.1% of the FP32 exponent footprint (42.7% for activations and 62.8% for weights).

***QM+QE*:** Figure 1d shows the total bitlength of the datatype for each tensor, including sign, mantissa, and exponent. It further amplifies the conclusions from above. Massive footprint reduction, significantly varying bitlength tensor to tensor justifying the fine-grained approach, and slightly increasing bitlength for some tensors learning period to period. To further emphasize the importance of the fine-grained approach we can look at the average and worst-case bitlengths. For instance, the worst-case activation tensor requires 11b while the average is less than 6b.

The variability of total, exponent and mantissa bitlenghts for weights and activations at the beginning of every epoch is shown in Figure 2. This figure shows that, while there are some tensors that share bitlenghts, for instance, weight exponents, most bitlengths vary wildly. The most important message from this graph though is that choosing datatypes is hard and complicated. If we want to squeeze as much of a reduction of footprint as possible, we need an automated method. It is impossible to guess the bitlengths in advance.

Cumulatively, on average, the datatype footprint is reduced by 3.86× (8.28 bits), 5.92× (5.40 bits) and 5.86× (5.46 bits) vs FP32 for weights, activations, and total footprint. Similarly, footprint reduction in comparison with BFloat16 is 1.93×, 2.96×, and 2.93× for weights, activations, and total, respectively. This is furhter emphasized in Figure 3. Finally, *QM+QE* datatype is 32% smaller than FP8.

***QM+QE* as a datatype selection advisor:** *QM+QE* quickly learns bitlengths that can be used to learn the per tensor datatypes to use for training the network, e.g., if we need to retrain the network we can use bitlenghts from the previous run as-is. The accuracy of such a run increased 0.2% of pre-



(a) Validation accuracy

(b) Mantissa bitlength

(c) Exponent bitlength

(d) Total bitlength

Figure 1: *QM* and *QE* on ResNet18/ImageNet throughout training: (a) Validation accuracy, (b) Weighted mantissa bitlengths with their spread, (c) Weighted exponent bitlengths with their spread, and (d) Weighted total bitlengths with their spread.

vious training with *QM+QE*. Similarly, bitlenghts learned in the first 5 epochs can be used with a small accuracy drop (0.7%). This capability is particularly useful given that industry is introducing a wide selection of leaner datatypes. It can aid or completely replace the current, manual, trial-and-error selection process allowing users to automatically benefit from the datatypes their hardware supports.

3.2 *BitWave*

Methods that do not interject into the training implementation, no matter how little, and that do not have any overhead are appealing. *BitWave* is such a method. *BitWave* monitors training progress as an outside observer adjusting the mantissa bitlength and exponent ranges accordingly: as long as the network is improving, *BitWave* will attempt to use a shorter mantissa (*BWM*) and to reduce the available exponent value range (*BWE*). The ideal scenario for *BitWave* is one where past observations of training progress are good indicators of forthcoming behavior. Fortunately, training is a long process based on *trial-and-error*, which may be forgiving for momentary lapses in judgement.

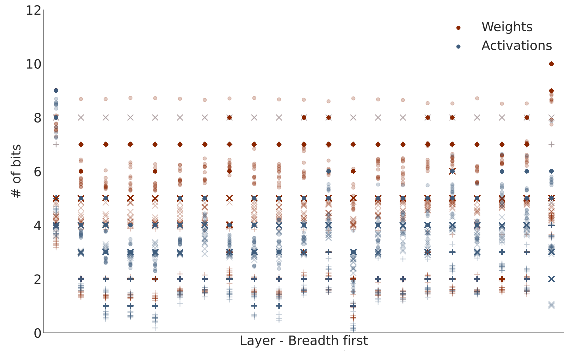


Figure 2: *Quantum Mantissa* and *Quantum Exponent* on ResNet18/ImageNet: mantissa ($+$), exponent ($\times$), and total ($\cdot$) bitlength datatypes of each tensor at the end of each epoch. Darker colors indicate multiple occurrences.

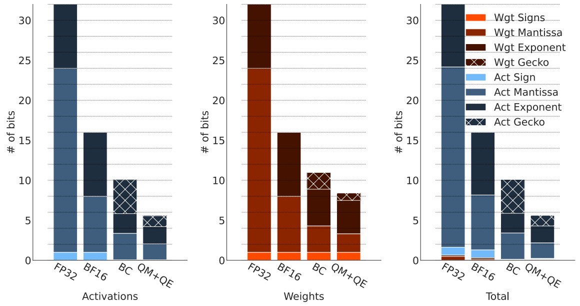
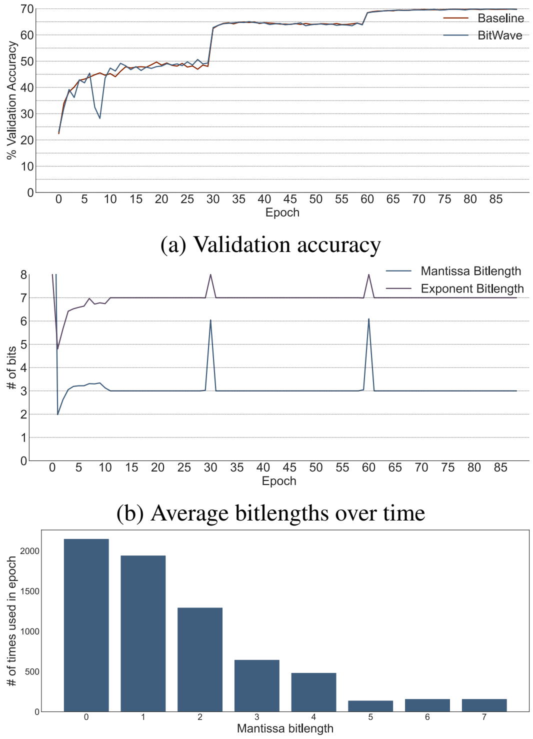


Figure 3: *Schrödinger's FP*: Relative training footprint of ResNet18 with FP32, BFloat16, SFP_{BW} and SFP_{Q}.

The main design decision that impacts how successful *BitWave* will be is the information to use as a proxy for training progress. *BitWave* should strike a balance between reducing bitlengths while avoiding over-clipping and hurting learning progress. We have experimented with several options and arrived at the following choices: 1) Using the slope of a simple linear regression over a history of the loss as a proxy for network progress, 2) observing training progress and adjusting bitlengths every batch, and 3) using the same bitlengths for the entire network.

While the loss improves over time, at batch granularity it exhibits non-monotonic (sometimes erratic) behavior. *BitWave* compensates for this by calculating a least squares regression (minimization of total sum of squared differences between predicted and history values) over a history of previous loss values. It uses the slope of the linear regression at each batch to smooth the non-monotonic behavior.

BitWave Mantissa (BWM): *BitWave* adjusts the mantissa



(a) Validation accuracy

(b) Average bitlengths over time

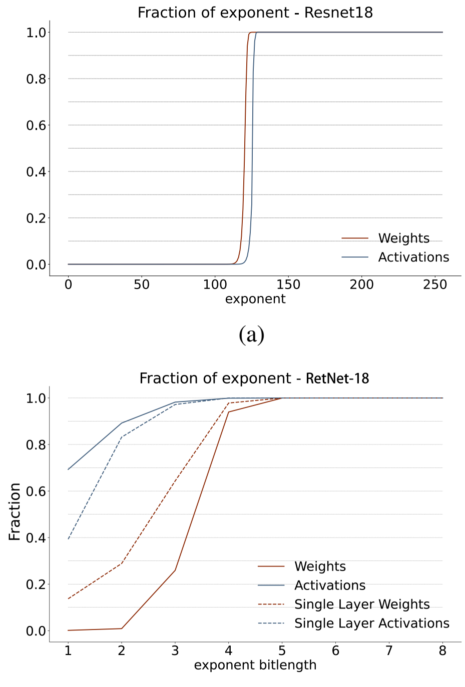
(c) Distribution of mantissa bitlengths

Figure 4: *BitWave* on ResNet18/ImageNet: (a) Validation accuracy throughout training, (b) Average mantissa and exponent bitlengths per epoch throughout training, (c) Distribution of *BitWave*'s mantissa bitlengths throughout the 5005 batches of epoch 5 of training.

length (unchanged, lower, or higher) by observing the slope of the linear regression. A negative slope indicates learning is improving, allowing for further mantissa trimming. A positive slope indicates no learning progress and *BitWave* responds by increasing mantissa bitlength. If the slope is within a small threshold T of 0.0, then *BitWave* keeps observing and does not alter the bitlength.

BitWave Exponent (BWE): Considering the range of FP32 exponents ($[-126, 127]$), *BitWave* adapts the range of values symmetrically by adjusting both limits. Exponents below the minimum are clamped to 0, whereas those above the maximum value saturate at that. This gradual change eventually reduces or increases the exponent bitlength. *BitWave* adjusts the exponent range (unchanged, lower, or higher) by examining the slope of the calculated linear regression. A negative slope (with a threshold T) is assumed to indicate improvement, allowing the range to shrink. A positive slope (with the same threshold T), indicates deteriorating learning so *BitWave* widens the exponent range.

Bitlength Selection Schedule: Similarly to *QM+QE*, *BitWave* produces non-deterministic datatypes due to its intrinsic fluctuations throughout training because of its heuristic nature. To avoid this non-determinism and provide usable



(a)

(b)

Figure 5: *Gecko* on ResNet18/ImageNet: (a) Cumulative distribution of exponent values. (b) Post-encoding cumulative distribution of exponent bitlength.

bitlengths for inference, *BitWave* fixes the mantissa bitlength and the exponent range after a few epochs of training, by calculating the average of all the bitlengths up to that point of training, as well as the average of the exponent range. *BitWave* then uses these averages for the rest of training. Experiments on the convergence of *BitWave* show that the networks converge to the same accuracies ($\pm 0.1\%$) whether the bitlengths are fixed or not, and there are evident benefits of creating deterministic inference-capable bitlengths.

Evaluation: Bitlengths and Accuracy: We report *BitWave*'s effect on footprint and accuracy during training of ResNet18. Figure 4a shows that validation accuracy is unaffected. Figure 4b shows that *BitWave* reduces mantissa bitlengths to 3b *on average* from baseline precision. However, mantissa bitlengths may vary slightly per batch as illustrated in the histogram (Figure 4c) of bitlengths used throughout a sample epoch. This shows that training sometimes requires the entire range whereas other times it only requires 0 bits. Across the training process, *BitWave* reduces the total mantissa footprint to 14.3% of baseline, and the total exponent footprint to 83.8%. While *BitWave* might miss bitlength reductions per layer and not reduce the exponent bitlength as much, it is non-intrusive and has no overhead.

3.3 Exponent: *Gecko* (+G)

Exponents are biased 8b integers under the BFloat16 and FP32 formats, and even narrower when optimized with *QE* or *BWE*. Despite this, all exponents per tensor are recorded

Table 1: $\mathrm{S}FP_{\mathrm{B}W}$ and $\mathrm{S}FP_{\mathrm{Q}}$: Validation metrics and total memory footprint reduction vs. FP32.

				$\mathrm{S}FP_{\mathrm{Q}}$						$\mathrm{S}FP_{\mathrm{B}W}$					
Network	**Task**	**Metric**	**FP32 Score**	**Score**	*QM*	*QE*	*QE+G*	*QM+QE*	*QM+QE +G*	**Score**	*BWM*	*BWE*	*BWE+G*	*BWM+BWE*	*BWM+BWE +G*
ResNet18	Image Classification	Accuracy	69.94	69.50	11.91×	2.322×	3.649×	5.857×	7.599×	69.95	6.961×	1.193×	3.235×	3.197×	5.539×
ResNet50	Image Classification	Accuracy	76.06	75.58	14.10×	2.277×	3.513×	6.192×	8.138×	75.80	7.385×	1.224×	2.688×	3.316×	5.254×
MobileNet V2	Image Classification	Accuracy	71.62	71.44	8.380×	2.073×	3.169×	4.818×	6.030×	71.35	7.320×	1.252×	2.390×	3.358×	4.931×
DLRM	Recommendation	Accuracy	79.42	79.39	14.58×	2.123×	2.724×	5.334×	6.191×	79.45	7.041×	1.167×	2.563×	4.113×	5.011×
ViT	Image Modeling Pre-training	Evaluation Loss	0.087	0.087	313.5×	5.947×	10.84×	13.23×	17.66×	0.092	151.68×	3.095×	7.426×	8.909×	9.741×
GPT-2	Language Modeling Fine-tuning	Perplexity	20.95	21.13	5.506×	1.822×	2.357×	3.345×	3.734×	21.12	4.159×	1.065×	2.165×	2.454×	3.469×
BERT - CoLA	Text Classification Fine-tuning - CoLA	Matthews Correlation	55.99	57.03	9.878×	1.996×	2.680×	4.362×	5.069×	56.11	6.864×	1.034×	2.085×	2.886×	4.452×
BERT - SST-2	Text Classification Fine-tuning - SST-2	Accuracy	93.23	91.97	15.31×	2.114×	2.806×	5.090×	5.976×	92.44	6.180×	1.032×	2.080×	2.789×	4.228×
BERT - MRPC	Text Classification Fine-tuning - MRPC	Accuracy	84.56	84.80	9.189×	1.988×	2.601×	4.252×	4.864×	84.59	3.574×	1.016×	2.082×	2.237×	3.114×
BERT - STS-B	Text Classification Fine-tuning - STS-B	Pearson	88.92	88.81	6.322×	1.821×	2.465×	3.544×	4.059×	89.11	3.501×	1.051×	2.078×	2.257×	3.071×
BERT - QQP	Text Classification Fine-tuning - QQP	Accuracy	90.71	90.30	11.69×	1.970×	2.585×	4.553×	5.278×	90.43	6.689×	1.119×	2.073×	3.022×	4.385×
BERT - MNLI	Text Classification Fine-tuning - MNLI	Matched Accuracy	83.87	84.16	9.786×	1.884×	2.470×	4.213×	4.856×	84.03	6.732×	1.069×	2.073×	2.936×	4.398×
BERT - QNLI	Text Classification Fine-tuning - QNLI	Accuracy	90.54	90.28	9.342×	1.903×	2.486×	4.175×	4.791×	90.54	6.552×	1.503×	2.071×	3.005×	4.340×
Geo Mean					11.94×	2.126×	2.972×	4.736×	5.637×		7.556×	1.228×	2.492×	3.185×	4.558×

using the same bit count. During training, these values tend to cluster heavily around a number, as shown in Figure 5a, which depicts the exponent distribution for ResNet18 after the 10th epoch. Leveraging this skewed distribution, we apply a variable-length, *lossless* encoding that adjusts bit usage to the actual value of each exponent, such as using only 2 bits for the value 3. This method involves subtracting a bias value from each exponent, allocating fewer bits to frequent values and more to rare ones, thus minimizing average bit usage. A 3b metadata field records the bitlength, shared across multiple exponents to minimize metadata overhead. We also observe that these values are spatially correlated, with proximal values often being similar.

Gecko encoding operates as follows: Given a tensor, *Gecko* groups values into sets of 8. Exponents are encoded as $E - bias$, where E is the original exponent and the bias is a fixed value; our experiments show that using 127 as the bias provides the best compression ratio. A leading 1 detector finds how many bits are needed for the largest exponent. That bitlength is recorded as metadata using 3 bits. All exponents of the group are stored using this bitlength. Using variable bitlengths for encoding values complicates random access, as it precludes direct computation of addresses in the tensor using their indices. However, deep learning workloads typically do not require random access to DRAM. Instead, blocking for data reuse leads to long sequential accesses to DRAM, which are conducive to the use of variable length containers. Variable bitlength encoding of values is common in quantization and memory compression methods (Han et al., 2016c;b; Lascorz et al., 2019).

Evaluation: Bitlength: We measure the number of bits needed for the exponents using *Gecko* during the training of ResNet18. Figure 5b reports the cumulative distributions of exponent bitlength for a batch across: 1) all layers and 2) a single layer, for weights and activations. After encoding, more than 90% of the weight exponents require 4b or fewer, while almost 90% of activation exponents require 2b or fewer. Across training, the compression ratios for weight and activation exponents are 0.60 and 0.38, respectively.

4 EVALUATION

We study the effects of *QM+QE* and *BWM+BWE* with, and without *Gecko* (*+G*). We *fully* train ResNet18, ResNet50 and MobileNet V2 on ImageNet, DLRM on Kaggle Criteo as well as pre-train ViT on Cifar10, finetune BERT on GLUE and GPT-2 on Wikitext 2, using an RTX3090/24GB with PyTorch v1.10. We implement *QM+QE* by modifying the loss function and adding the gradient calculations for the per tensor parameters. We simulate *BWM+BWE* in software. For both methods, we *faithfully* emulate *all* bitlength arithmetic effects by truncating the mantissa bits and encoding/decoding exponents at the boundary of each layer using PyTorch hooks and custom layers. We measure *Gecko*'s effects in software via hooks. These changes allow us to measure the effects our methods have on traffic and accuracy. The following discussion is illustrated for ResNet18 in Figure 3 and shown in full detail for all networks in Table 1.

4.1 Memory Footprint Reduction

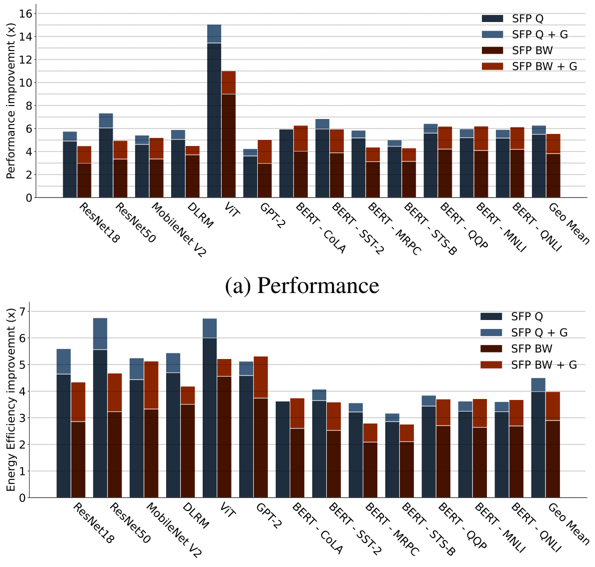
First, we report cumulative memory footprint reduction and validation accuracy in comparison with FP32 in Table 1. Our compression techniques excel at reducing footprint, with little effect on accuracy.

QM+QE reduces the total training footprint by $3.35\times$ to $13.23\times$ with an average of $4.74\times$. While *QM+QE* works great on exponent as well, it is exceptionally good at compressing mantissas. With the addition of *+G*, the benefits further extend to $3.73\times$ to $17.66\times$ with an average of $5.64\times$. *BWM+BWE* on the other hand reduces the total training footprint by $2.24\times$ to $8.91\times$ with an average of $3.19\times$ without, and $3.07\times$ to $9.74\times$ with an average of $4.56\times$ with *+G*, respectively. While *BWM+BWE* provides great compression rate for mantissas, it is less effective for exponents. The addition of *+G* recovers most of the compression gap. In the end, *QM+QE* outperforms *BWM+BWE* in every single case. However, this comes with the *slightly* larger overhead.

The optional $+G$ boosts the compression rate by 19% and 43% on top of *QM+QE* and *BWM+BWE*, respectively. It works far better for *BWM+BWE* because the method greatly removes outliers by focusing on exponent range, and helps it recover almost all of the exponent compression gap. This comes at the cost of variable tensor sizes, and therefore inability of random memory accesses to the off-chip memory. Fortunately, training only requires sequential access to off-chip memory, and sequential/strided/random accesses to on-chip memory which are fully supported by our design.

Table 2: *Quantum Integer*: ResNet18 Validation Accuracy and total memory reduction vs. FP32.

	FP32	*Quantum Integer*	
Network	**Accuracy**	**Accuracy**	**Footprint Reduction**
ResNet18	69.94	69.15	6.21×



(a) Performance

(b) Energy Efficiency

Figure 6: Comparison with FP32.

4.2 Quantization Alternatives

Alternative quantization approaches require selecting a datatype for training and sticking to it. The choice practically boils down to FP32, Bfloat16, and FP8. Table 1 shows memory reduction in comparison with FP32. Assuming that the network converges with the smaller datatype, Bfloat16 would always reduce the footprint by 2× and FP8 by 4×. Every single combination of our methods/networks outperforms FP32 and BFloat16 by a significant margin.

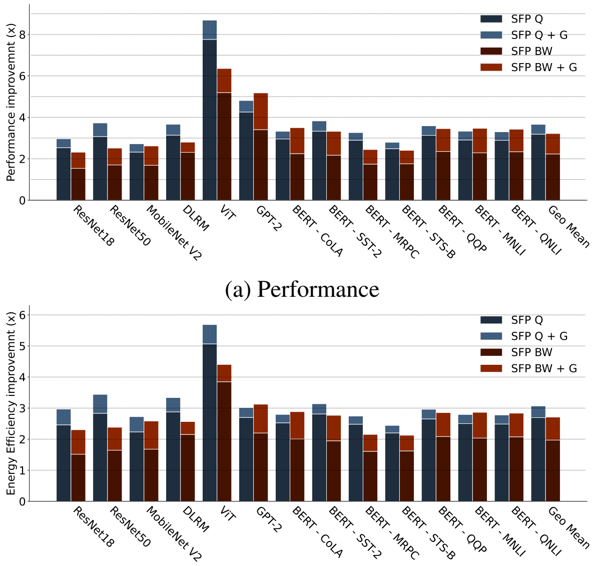
Furthermore, *QM*+*QE* produces a 16% smaller footprint than FP8 with GPT-2 being the only network where FP8 wins. With +*G*, *QM*+*QE*'s advantage increases to 29%.

The case for *BWM*+*BWE* vs FP8 isn't as clear cut, outperforming FP8 in all non CNNs. With +*G*, *BWM*+*BWE*, on average, produces a 12% smaller footprint than FP8.

Another key benefit of our methods is that they are *adaptable*. Choosing FP8 is risky, since the results of training is only evident at the end. Micikevicius et al. (2022) show that FP8 is a good choice for many networks, but they also note that there are architectures for which it is not sufficient. Our method provides a greater certainty of success, whilst obtaining better footprint.

4.3 *Quantum Integer* (*QI*): Fixed-Point Datatype

For some models, fixed-point training is possible. While our main goal is to learn the optimal *floating-point* datatypes, *QM* can easily be adapted to learn optimal fixed-point datatypes. One common way to train for fixed-point inference is by representing the activations in fixed point during training and to use integer arithmetic during the forward pass. The only modification we need is to switch out Eq. 5 for one that represents fixed-point. Other aspects of *QM* stay the same, while exponent is not used.



(a) Performance

(b) Energy Efficiency

Figure 7: Comparison with BFloat16.

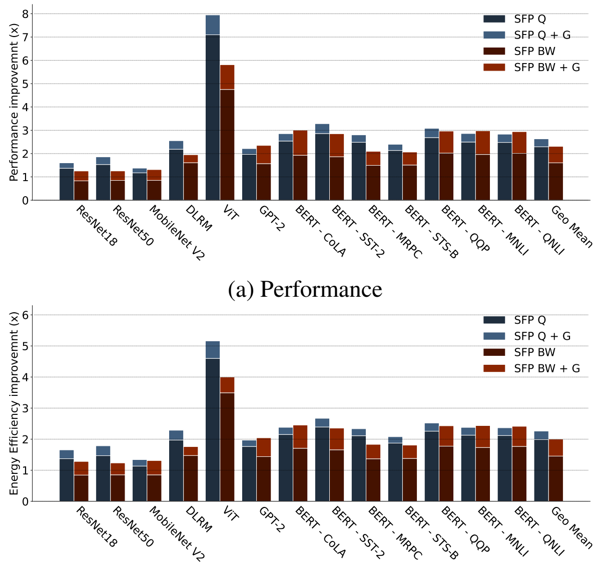
We present the footprint reduction and accuracy effect of the resulting *QI* by showing results with ResNet18 on ImageNet in Table 2. This simple, yet effective, modification learns the per-tensor optimal bitlengths for uniform quantization training with minimal accuracy cost. This is also a good choice for training when we are confident that the task the network is solving can be done in low bitlength fixed-point. The *QI* behavior is very similar to *QM*.

5 Performance and Energy Efficiency

We evaluate execution time and energy efficiency with *Schrödinger's FP* for all networks listed in Table 1. Figures 6, 7 and 8 report execution time and energy improvements in comparison with FP32, BFloat16 and FP8 baselines, respectively. The baseline datatype is used for weight updates and gradients. We further assume that all the networks can be trained to baseline accuracy with FP8 and BFloat16. However, this is not given.

5.1 Hardware Evaluation Methodology

We assess the execution time and energy efficiency by integrating *Schrödinger's FP* units into a hardware accelerator that reflects state-of-the-art designs. The accelerator has 8k



(a) Performance

(b) Energy Efficiency

Figure 8: Comparison with FP8.

units (each capable of performing 4 MACs per cycle on the baseline datatype), and a 500MHz clock for a peak compute bandwidth of 16TFLOPS. It uses 8 channels of LPDDR4-3200 DRAM memory and 32MB of on-chip buffers.

Our evaluation relies on an analytical model of the accelerator. We use CACTI (Muralimanohar & Balasubramonian) to model on-chip structures and DRAMSIM3 (Li et al., 2020) to estimate the time and energy for off-chip memory accesses. The hardware units are implemented in Verilog, synthesized using the Synopsys Design Compiler (Synopsys), and are laid out with Cadence Innovus (Cadence). The area overhead for the compressor and decompressor proves negligible, taking only 0.36% of the total accelerator area, excluding on-chip memory.

Power estimation is performed in Innovus using traces from a representative sample to accurately model signal activity. Appendix A describes our hardware units and the accelerator. Appendix B explains in more detail the evaluation methodology and the analytical model used for execution time and energy efficiency estimation.

5.2 Hardware Evaluation

We first compare *Schrödinger's FP* with the FP32 and BFloat16 baselines. FP32 has been the safe choice for training. Similarly, BFloat16 is another common choice for training. Figures 6 and 7 show that all versions of *Schrödinger's FP* greatly outperform both the FP32 and BFloat16 baselines in performance and energy efficiency, for every single network.

FP8 is a riskier datatype for training. For some networks and tasks it is sufficient, but for others it is not. Figure 8 shows that both *QM+QE* and *QM+QE+G* noticeably outperform FP8 in both performance and energy efficiency, for all networks. The figure also shows that *BWM+BWE* generally outperforms FP8, having a better average performance and energy efficiency. The outliers where results would be better with FP8 are all for ImageNet CNNs. MobileNet V2, one of the three outliers, can not be trained in FP8 because doing so introduces a noticeable accuracy loss. However, *BWM+BWE* can be used as shown in Table 1. Finally, our methods coupled with *Gecko* outperform FP8 on every network, both in performance and energy efficiency.

In general, as the baseline gets smaller, training becomes riskier, and the margins by which all *Schrödinger's FP* methods outperform become smaller. At the boundary where the accuracy starts to degrade, our methods are still significantly faster and more energy efficient.

QM+QE outperforms *BWM+BWE* across all networks. However, when used with *Gecko* the difference between our two methods is not as pronounced. *QM+QE+G* is still better but the difference is smaller and in some cases it even reverses (e.g. GPT-2).

Finally, we assumed that all benefits from smaller datatypes come from off-chip memory transfers. For accelerators where compute can also be made more efficient through using smaller, spatially composable or bit serial compute units, improvement with our methods would be even greater.

6 Conclusion

We introduced methods that dynamically adapt the bitlengths and containers used for floating-point values during training. The different distributions of the exponents and mantissas led us to tailored approaches for each. We target the largest contributors to off-chip traffic during training for both activations and weights. In addition, in the case where fixed-point training is preferred, we showed the effectiveness of our approach to determine the best containers used for fixed-point values during training. To our knowledge, this is the first work that demonstrates how to: (1) *determine* and (2) *continuously adjust* the memory containers (how many bits should be used when storing floating-point mantissas and exponents in memory), and to do so (3) *on-the-fly*, for the purpose of (4) making *training* itself faster and/or more energy efficient. There are several directions for improvements and further exploration including expanding the methods to also target the gradients and refining the underlying policies they use to adapt mantissa lengths. Regardless, this work has demonstrated that the methods are effective and superior to using fixed preselected datatypes. The key advantages of our methods are: 1) they are dynamic and adaptive, 2) they do not modify the training algorithm, 3) they will naturally extend to future algorithms without modifications and 4) they take advantage of value content.

References

Awad, O. M., Mahmoud, M., Edo, I., Zadeh, A. H., Bannon, C., Jayarajan, A., Pekhimenko, G., and Moshovos, A. Fpraker: A processing element for accelerating neural network training. In *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*, pp. 857–869. ACM, 2021. doi: 10.1145/3466752.3480106. URL https://doi.org/10.1145/3466752.3480106.

Cadence. Innovus implementation system. https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/hierarchical-design-and-floorplanning/innovus-implementation-system.html.

Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D. D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., Heinecke, A., Dubey, P., Corbal, J., Shustrov, N., Dubtsov, R., Fomenko, E., and Pirogov, V. O. Mixed precision training of convolutional neural networks using integer operations. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=H135uzZ0-.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. URL https://arxiv.org/abs/2010.11929.

Drumond, M., Lin, T., Jaggi, M., and Falsafi, B. Training DNNs with hybrid block floating point. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 451–461, USA, 2018. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=3326943.3326985.

Evans, R. D. and Aamodt, T. AC-GC: Lossy activation compression with guaranteed convergence. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=MwFdqFRxIF0.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. EIE: efficient inference engine on compressed deep neural network. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, pp. 243–254, 2016a.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. Eie: Efficient inference engine on compressed deep neural network. In *Intl' Symp. on Computer Architecture*, 2016b.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016c. URL http://arxiv.org/abs/1510.00149.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

HewlettPackard. CACTI. https://github.com/HewlettPackard/cacti.

Horowitz, M. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014. doi: 10.1109/ISSCC.2014.6757323.

Huang, X., Shen, Z., Li, S., Liu, Z., Xianghong, H., Wicaksana, J., Xing, E., and Cheng, K.-T. SDQ: Stochastic differentiable quantization with mixed precision. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9295–9309. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/huang22h.html.

Jain, A., Phanishayee, A., Mars, J., Tang, L., and Pekhimenko, G. Gist: Efficient data encoding for deep neural network training. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, pp. 776–789, Piscataway, NJ, USA, 2018. IEEE Press. ISBN 978-1-5386-5984-7. doi: 10.1109/ISCA.2018.00070. URL https://doi.org/10.1109/ISCA.2018.00070.

Judd, P., Albericio, J., Hetherington, T., Aamodt, T. M., Jerger, N. E., and Moshovos, A. Proteus: Exploiting numerical precision variability in deep neural networks. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pp. 23:1–23:12, New York, NY, USA, 2016a. ACM. ISBN 978-1-4503-4361-9. doi: 10.1145/2925426.2926294. URL http://doi.acm.org/10.1145/2925426.2926294.

Judd, P., Albericio, J., Hetherington, T., Aamodt, T. M., Jerger, N. E., and Moshovos, A. Proteus: Exploiting numerical precision variability in deep neural networks. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pp. 23:1–23:12, New York, NY, USA, 2016b. ACM. ISBN 978-1-4503-4361-9. doi: 10.1145/2925426.2926294. URL http://doi.acm.org/10.1145/2925426.2926294.

Kalamkar, D. D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., Yang, J., Park, J., Heinecke, A., Georganas, E., Srinivasan, S., Kundu, A., Smelyanskiy, M., Kaul, B., and Dubey, P. A study of BFLOAT16 for deep learning training. *CoRR*, abs/1905.12322, 2019. URL http://arxiv.org/abs/1905.12322.

Köster, U., Webb, T. J., Wang, X., Nassar, M., Bansal, A. K., Constable, W. H., Elibol, O. H., Gray, S., Hall, S., Hornof, L., Khosrowshahi, A., Kloss, C., Pai, R. J., and Rao, N. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 1740–1750, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. URL http://dl.acm.org/citation.cfm?id=3294771.3294937.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Lascorz, A. D., Sharify, S., Edo, I., Stuart, D. M., Awad, O. M., Judd, P., Mahmoud, M., Nikolić, M., Siu, K., Poulos, Z., and Moshovos, A. Shapeshifter: Enabling fine-grain data width adaptation in deep learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 52, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369381. doi: 10.1145/3352460.3358295. URL https://doi.org/10.1145/3352460.3358295.

Li, S., Yang, Z., Reddy, D., Srivastava, A., and Jacob, B. Dramsim3: A cycle-accurate, thermal-capable dram simulator. *IEEE Computer Architecture Letters*, 19(2):106–109, 2020. doi: 10.1109/LCA.2020.2973991.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *CoRR*, abs/1609.07843, 2016. URL http://arxiv.org/abs/1609.07843.

Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., García, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Muralimanohar, N. and Balasubramonian, R. Cacti 6.0: A tool to understand large caches.

Naumov, M., Mudigere, D., Shi, H. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., Chen, W., Rao, V., Jia, B., Xiong, L., and Smelyanskiy, M. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019. URL https://arxiv.org/abs/1906.00091.

Nikolić, M., Mahmoud, M., and Moshovos, A. Characterizing sources of ineffectual computations in deep learning networks. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 86–87, 2018. doi: 10.1109/IISWC.2018.8573509.

Nikolić, M., Hacene, G. B., Bannon, C., Lascorz, A. D., Courbariaux, M., Bengio, Y., Gripon, V., and Moshovos, A. Bitpruning: Learning bitlengths for aggressive and accurate quantization, 2020.

NVIDIA. Training with mixed precision. https://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/index.html.

Qian Zhang, S., McDanel, B., and Kung, H. T. Fast: Dnn training under variable precision block floating point with stochastic rounding. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 846–860, 2022. doi: 10.1109/HPCA53966.2022.00067.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Rouhani, B., Zhao, R., Elango, V., Shafipour, R., Hall, M., Mesmakhosroshahi, M., More, A., Melnick, L., Golub, M., Varatkar, G., Shao, L., Kolhe, G., Melts, D., Klar, J., L'Heureux, R., Perry, M., Burger, D., Chung, E., Deng, Z., Naghshineh, S., Park, J., and Naumov, M. With shared microexponents, a little shifting goes a long way, 2023a.

Rouhani, B. D., Zhao, R., More, A., Hall, M., Khodamoradi, A., Deng, S., Choudhary, D., Cornea, M., Dellinger, E., Denolf, K., Dusan, S., Elango, V., Golub, M., Heinecke, A., James-Roxby, P., Jani, D., Kolhe, G., Langhammer, M., Li, A., Melnick, L., Mesmakhosroshahi, M., Rodriguez, A., Schulte, M., Shafipour, R., Shao, L., Siu, M., Dubey, P., Micikevicius, P., Naumov, M., Verrilli, C., Wittig, R., Burger, D., and Chung, E. Microscaling data formats for deep learning, 2023b.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575 [cs]*, September 2014. arXiv: 1409.0575.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. MobileNetV2: inverted residuals and linear bottlenecks. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2018.

Sun, X., Wang, N., Chen, C.-Y., Ni, J., Agrawal, A., Cui, X., Venkataramani, S., El Maghraoui, K., Srinivasan, V. V., and Gopalakrishnan, K. Ultra-low precision 4-bit training of deep neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1796–1807. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/13b919438259814cd5be8cb45877d577-Paper.pdf.

Synopsys. Design Compiler. http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. HAQ: hardware-aware automated quantization. *CoRR*, abs/1811.08886, 2018a. URL http://arxiv.org/abs/1811.08886.

Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. Training deep neural networks with 8-bit floating point numbers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 7686–7695, Red Hook, NY, USA, 2018b. Curran Associates Inc.

Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., and Keutzer, K. Mixed precision quantization of convnets via differentiable neural architecture search. *CoRR*, abs/1812.00090, 2018. URL http://arxiv.org/abs/1812.00090.

Yang, H., Duan, L., Chen, Y., and Li, H. {BSQ}: Exploring bit-level sparsity for mixed-precision neural network quantization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=TiXl51SCNw8.

Zhang, X., Liu, S., Zhang, R., Liu, C., Huang, D., Zhou, S., Guo, J., Guo, Q., Du, Z., Zhi, T., and Chen, Y. Fixed-point back-propagation training. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2327–2335, 2020. doi: 10.1109/CVPR42600.2020.00240.