arXiv:2405.18710v2 [cs.LG] 25 Mar 2025

To FP8 and Back Again: Quantifying Reduced Precision Effects on LLM Training Stability

Joonhyung Lee [1] Jeongin Bae [1] Byeongwook Kim [1] Se Jung Kwon [1] Dongsoo Lee [1]

Abstract

The massive computational costs associated with large language model (LLM) pretraining have spurred great interest in reduced-precision floating-point representations to accelerate the process. As a result, the BrainFloat16 (BF16) precision has become the *de facto* standard for LLM training, with hardware support included in recent generations of accelerators. This trend has gone even further in the latest processors, where FP8 has recently been introduced. However, prior experience with FP16, which was found to be less stable than BF16, raises concerns as to whether FP8, with even fewer bits than FP16, can be a cost-effective option for LLM training. We argue that reduced-precision training schemes must have similar training stability and hyperparameter sensitivities to their higher-precision counterparts in order to be cost-effective. However, we find that currently available methods for FP8 training are not robust enough to allow their use as economical replacements. This prompts us to investigate the stability of reduced-precision LLM training in terms of robustness across random seeds, learning rates, and datasets. To this end, we propose new evaluation techniques and a new metric for quantifying loss landscape sharpness in autoregressive language models. By simulating incremental bit reductions in floating-point representations, we analyze the relationship between representational power and training stability with the intent of aiding future research into the field.

1. Introduction

Conversational large language models (LLMs), such as ChatGPT (OpenAI, 2024), Gemini (Team, 2024a;b), Claude (Anthropic, 2024), DeepSeek (DeepSeek-AI, 2024) and HyperCLOVA (Yoo et al., 2024), have captured the imagination of both academics and the public at large with their ability to communicate fluently with humans in natural language. However, these models require unprecedented amounts of computation to train, which has engendered interest in methods to improve their training efficiency.

A popular method of improving computational performance is to reduce the bit count of the floating-point representations used for training (Wang et al., 2018; Sun et al., 2020; Peng et al., 2023). Because reading memory is the main bottleneck in modern processors, a problem known as the "memory wall" (Wulf & McKee, 1995; Kim et al., 2023), reducing the number of bits that each floating-point number uses can accelerate the computation in proportion to the amount of memory reduced. For example, in processors that support it, computations in BrainFloat16 (BF16) (Kalamkar et al., 2019) can have double the maximum throughput of single precision FP32. Furthermore, the FP32 data type, the highest precision data type used in deep learning, has only half the bits of FP64, the most widely used floating-point data type in scientific computing. The current best practice for LLM training is to use BF16 for most of the LLM training computation, with some sensitive portions, such as layer normalization (Ba et al., 2016), carried out in FP32.

As a natural extension of this development, 8-bit floating-point (FP8) (Wang et al., 2018; Sun et al., 2019; Micikevicius et al., 2022; Peng et al., 2023) and even 4-bit floating-point (Sun et al., 2020) data formats have been proposed to accelerate training even further. However, the naïve application of FP8 to LLM training is unstable and requires additional techniques to become viable (Shen et al., 2024). While several methods have been proposed to stabilize training LLMs with FP8, relatively little attention has been paid to quantifying the decrease in stability compared to mixed-precision BF16 training.

Cost reduction is the motivation behind the use of FP8 and other reduced-precision training schemes as there is no intrinsic benefit in training models using smaller data types. Therefore, our concern is not whether LLM pretraining with FP8 is possible but whether it is profitable. For cost savings to be realized, the time per training step must be reduced while the number of training steps is kept to a similar number. Training stability is thus a crucial factor

[1]NAVER Cloud, Seongnam, Republic of Korea.

for cost-effective LLM training, considering that additional hyperparameter searches and restarts from training failures can outweigh any gains from raw compute acceleration.

Previous experience with training LLMs in FP16 raises further concerns. Teams that have trained LLMs have found that even when gradient scaling and other techniques are applied, the FP16 data type, which has five exponent bits, is much less stable for LLM training than BF16, which has eight exponent bits as in FP32. This raises doubts as to whether FP8, which has even fewer bits than FP16, is a practical option for real-world LLM training.

We motivate our line of inquiry with some surprising findings from experiments on the nanoGPT (Karpathy, 2022) codebase, an open-source implementation of GPT-2 pretraining, where we found that even the current best practice of mixed-precision BF16 can introduce training instabilities. When we compared BF16 and TensorFloat32 (TF32) runs, where we ran training for 5% of the original configuration, we found that the BF16 models diverged for 18 of 188 runs, or approximately 10% of all cases, despite using the same configurations as the default run. In contrast, no cases of loss divergence were found for the 70 TF32 models trained using different random seeds. We use the TF32 data type because NVIDIA GPUs do not offer FP32 tensor cores.

This is surprising in light of the fact that most recent LLMs are trained with mixed precision BF16 without a comparison with training on TF32, which has three additional mantissa bits. However, a loss divergence rate of approximately 10% at only 5% of training indicates that even standard BF16 mixed-precision training may add non-trivial instability. If even mixed-precision BF16 can cause instabilities, the effects of using even fewer bits should be investigated further.

We make the following contributions in our work:

- We analyze the narrower hyperparameter space that emerges from reducing precision by clipping mantissa bits to simulate intermediate-bit floating-point representations and find greater instability in the model when exposed to higher learning rates or “dirtier” data.
- We propose a metric for quantifying the loss landscape sharpness of models that can predict when training divergence will occur. As even the removal of mantissa bits has a destabilizing effect on LLM training, we use our metric to predict loss divergences even when the loss curve itself has not yet diverged.

To prevent misunderstanding, we also clarify on points we do **not** make. We do not argue that LLM pretraining with FP8 is impractical, only that there must be a deeper investigation into whether proposed methods are economical. While various techniques for FP8 training (Fishman et al.,

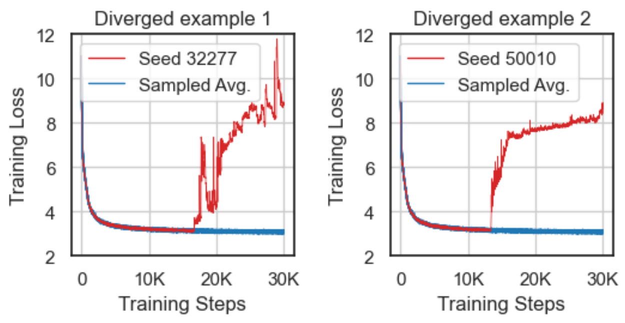

Figure 1. We show an example case of loss divergence on nanoGPT when using the same configurations as the default run except for the random seed. The blue lines indicate the average losses obtained for eight training runs that did not diverge. Of the 188 random seeds that were tested, 18 were found to diverge. As full pretraining requires over 4 days on a single node with 8 A100 GPUs, even for BF16, we perform early stopping at 30K steps, or 5% of the original training steps, requiring approximately 4 hours for a BF16 run and 8 hours for a TF32 run per A100 node with 8 GPUs. Because we only run 5% of the original training, we suspect that the measured divergence rate of approximately 10% underestimates the true rate of training loss divergence.

2024; Peng et al., 2023; Xi et al., 2025) and even FP4 training (Wang et al., 2025) have been proposed, what remains unclear is whether these methods results in a cost reduction when considering factors other than raw compute speed.

During training, a single loss divergence makes redundant the GPU time from the last checkpoint up to the idle period during the restart, possibly canceling out any gains from the accelerated computation. Also, during actual FP8 training, we cannot know what the training curve would be had BF16 been used instead. If the model requires additional steps to converge to the same point, this would further increase the cost. Even if training works reasonably well on standardized datasets and configurations, practitioners wish to know whether these findings are generalizable. We seek not to discredit any particular method but to show how a new training method can prove itself robust and trustworthy.

2. Related work

2.1. DeepSeek V3

As we were preparing this work, DeepSeek V3 (DeepSeek-AI, 2024) has gained immense popularity as a open-source SOTA model that is capable of rivaling closed-source competitors such as ChatGPT o1. Notably, it proposes a sophisticated FP8 quantization scheme with fine-grained FP8 dynamic scaling with a group size of 128 and accumulation to FP32 for every 4 WGMMA instructions to compensate for the 14-bit accumulation of NVIDIA FP8 tensor cores. This raises a question. Had previously proposed FP8 training schemes been sufficiently stable to train LLMs cost effectively, it would not be necessary to write customized kernels. For example, in none of the previous works (Xi et al., 2025; Fishman et al., 2024; Peng et al., 2023; NVIDIA, 2023) on FP8 training was a scaling factor granularity of 128 proposed, with per-row scaling being the finest granularity used. Furthermore, prior to DeepSeek V3, the backward pass was implemented using E5M2, with it being the first work that we are aware of to apply E4M3 to the backward pass as well as the forward pass.

We take the unconventional FP8 implementation in the work as confirmation of our claim that previous attempts at FP8 training had not gained economical viability. Furthermore, we argue that new implementations of FP8 should undergo rigorous analysis on whether they truly are equivalent to BF16 training, and not overfit to the architectures and hyperparameters used for well-known tasks.

2.2. Training stability

Analyzing training instability in LLM pretraining directly is impractical due to the massive costs involved. Instead, smaller language models must be used as proxies. Wortsman et al. (2024) explore the robustness of smaller language models as a proxy for LLM pretraining instability. They find that small models using a larger learning rate show similar instability patterns as larger models, such as the growth of attention layer logits and the divergence of the output logits from the log probabilities. They also explore both the causes of numerical instability in LLM training and mitigating strategies such as applying query/key (QK) normalization (Dehghani et al., 2023).

Keskar et al. (2017) explore the sharpness of loss landscapes during large-batch neural network training, finding that larger batch sizes prevent the model from reaching flat regions of the loss landscape and causing performance degradation due to the inability to escape local minima. Of most significance to our work, they propose a metric for calculating loss landscape sharpness, which we adapt for LLMs as a proxy for training instability.

Fishman et al. (2024) go further, discovering that Llama 7B models trained in FP8 begin to diverge after 200B tokens of training, supporting our claim that reduced-precision methods induce hidden instabilities that may emerge only later in training. They identify SwiGLU (Shazeer, 2020) as a source of massive activations (Sun et al., 2024) and apply additional scaling factors to reduce the instability. However, it remains unclear whether the proposed method is equivalent to BF16 training or are simply stable enough for the experiments involved.

2.3. Reduced-precision processors

To improve throughput on computationally intensive matrix multiplication tasks, recently developed processors have been equipped with specialized hardware units such as systolic arrays for TPUs (Jouppi et al., 2017) and tensor cores (NVIDIA, 2020) for NVIDIA GPUs. These processors can improve throughput by an order of magnitude. For example, on the H100, the peak BF16 GEMM throughput on tensor cores is 989.4 TFLOPS, compared to 133.8 TFLOPS on CUDA cores (NVIDIA, 2022).

However, the number of multiplexer circuits required for the barrel shifter of an n-bit floating-point unit is $n \log_2 n$ (Kroening & Strichman, 2008), which incentivizes using smaller floating-point representations. As a result, many mixed-precision techniques perform computationally intensive matrix multiplication in BF16 while preserving sensitive portions of the model, such as the master weights and residual path activations, in FP32.

2.4. Hybrid FP8

The adoption of the hybrid E4M3/E5M2 formats for neural networks (Micikevicius et al., 2022) in recent generations of processors, such as the NVIDIA H100 and the Intel Gaudi v2, has spurred interest in stable FP8 training. The hybrid FP8 format, where E4M3 is used for the forward pass for its greater resolution, and E5M2 is used for the backward pass for its greater range, was first proposed by Wang et al. (2018) as a means to accelerate neural networks.

Sun et al. (2019) built on this work to propose various techniques for stabilizing training, such as stochastic rounding, chunk-based accumulation, and Kahan summation during the optimizer update. However, the number of techniques that can be used in practice is limited by whether the technique in question can be applied without slowing the computation. As currently available NVIDIA GPUs, by far the accelerators with the greatest adoption, do not support these techniques natively, the overhead caused by the software-based implementations cancels out any gains from the reduced precision computations.

2.5. MS-AMP

Introduced in Peng et al. (2023), MS-AMP is an automatic mixed-precision package to utilize FP8 that offers multiple optimization levels to allow for differing model sensitivities when applying reduced precision to the computations and communications of neural network training. Our experiments use the O1 optimization level of MS-AMP, which performs GEMM computations in FP8 while reducing the weights to half-precision and uses FP8 states for the weight gradients and all-reduce communications. MS-AMP offers additional optimizations for the optimizer buffer in level O2

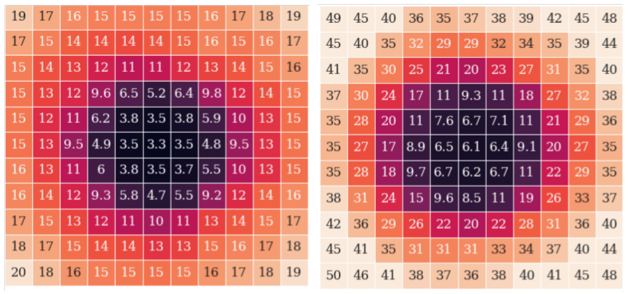

Figure 2. Loss landscape diagrams for Llama 120M E8M3 at 5K steps (left) and 10K steps (right). Even during loss divergence, the loss landscape visualized using the method in Li et al. (2018) appears smooth, motivating our introduction of a new loss landscape sharpness metric. The validation loss is shown for each point of the loss landscape.

optimization and for distributed communication in level O3 optimization, but we use only the most basic scheme so as to verify the effects of the least invasive modifications.

3. Methods

We seek to answer whether sub-BF16 training is competitive with standard mixed-precision BF16 training from a cost-effectiveness point of view. To be cost-effective, reduced-precision training schemes must have minimal increases in training instability and changes to hyperparameters. To better analyze the effect of reduced precision on training stability, we aim to quantify the effects of gradually reducing floating-point representations bit by bit. Hopefully, analyzing the intermediate bit representations will better illuminate the interaction between bit width and training stability.

Our intermediate-bit floating-point experiments use the TinyLlama (Zhang et al., 2024) repository, which implements the widely used Llama (Touvron et al., 2023a;b) architecture. TinyLlama is an open-source implementation of Llama pretraining that uses a mix of the SlimPajama (Soboleva et al., 2023) and StarCoder (Li et al., 2023) datasets for training. It also includes performance optimizations such as Flash Attention v2 (Dao, 2023). We use the default learning rate of $lr = 4e-4$, global batch size 512, and the same learning rate schedule as in the original code. The 120M models use a sequence length of 2048, while the 7B models use a sequence length of 4096.

3.1. Sharpness metric

To better investigate the model state when loss divergence occurs, we attempted to visualize the loss landscape of the Llama models with the method proposed by Li et al. (2018). However, as shown in Figure 2, we found that even when the model is clearly in the process of loss divergence, the generated visualizations remain smooth. We emphasize

```
def forward(x, w):
    # x: input, w: weight, out: output
    save_for_backward(x, w)
    masked_x = reduce_precision(x)
    masked_w = reduce_precision(w)
    out = F.linear(masked_x, masked_w)
    masked_out = reduce_precision(out)
    return masked_out
```

Figure 3. PyTorch-like pseudocode for the forward pass.

that we do not seek the loss landscape sharpness *per se*, but a proxy to provide a quantitative measurement of the underlying training instability.

Because of this, we propose an alternative loss landscape sharpness metric that is more suitable for autoregressive models, based on the one proposed in Keskar et al. (2017). We empirically confirm that it is a useful indicator of training instability in the following sections. The main difference between the original metric and our version is that we use the logit of the last token instead of the model input for the calculation. This is because adding noise to the embeddings in a language model has different implications compared to adding noise to input images in a vision model. We also do not apply random projection as in Keskar et al. (2017) to reduce the stochasticity of the measurement.

Instead of searching the input space as in Keskar et al. (2017), we apply the search algorithm to the logit space of the last token. Searching the logit space has the additional advantage that the forward pass of the model need only be performed once for each measurement, significantly reducing the computational cost. The logit of the last token was chosen because it is not computationally feasible to optimize for the entire output space. Also, due to the autoregressive character of decoder-only Transformer models (Vaswani et al., 2017; Brown et al., 2020), the last token is the only one to receive inputs from all other tokens.

Definition Let $\boldsymbol{y} \in \mathbb{R}^{s \times v}$ be the output logit for an autoregressive model of sequence length s and vocabulary size v. Then, for $\boldsymbol{y}_i$, the output logit at sequence position $i \in \{1, 2, ..., s\}$, and one-vector $\mathbf{1}_v \in \mathbb{R}^v$, we define a constraint set $\mathcal{C}_\epsilon$ at $i = s$ such that

$$\mathcal{C}_\epsilon \in \{\boldsymbol{z}_s \in \mathbb{R}^v : -\epsilon(|\boldsymbol{y}_s|+\mathbf{1}_v) \leq \boldsymbol{z}_s \leq \epsilon(|\boldsymbol{y}_s|+\mathbf{1}_v)\}. \quad (1)$$

Given $\boldsymbol{y}_s \in \mathbb{R}^v, \epsilon > 0$, and noise vector z_s, the loss landscape sharpness ϕ_ϵ for loss function f can be defined as

$$\phi_\epsilon := \frac{max_{\boldsymbol{z}_s \in \mathcal{C}_\epsilon} f(\boldsymbol{y}_s + \boldsymbol{z}_s) - f(\boldsymbol{y}_s)}{1 + f(\boldsymbol{y}_s)} \times 100. \quad (2)$$

The proposed metric can best be thought of as the relative magnitude of the largest loss spike on the logit within the provided bounds. The bounds are set to be the logit magnitudes plus one multiplied by ϵ. The largest spike in the vicinity of the logits is found using the L-BFGS-B algorithm (Liu & Nocedal, 1989), using the SciPy (Virtanen et al., 2020) implementation with the output logit set as the starting point of the search. We set $\epsilon = 5e{-}4$ for all our experiments following Keskar et al. (2017). However, a hyperparameter sweep on ϵ shows that the general trend is unaffected by the value of ϵ, only the sharpness value magnitudes. The results are shown in Table 2 of the Appendix.

3.2. Masking

Our experiments use a simplified method of reducing floating-point precision to achieve reasonable throughput. To simulate removing exponent bits, we threshold the values to the minimum and maximum absolute values possible with the given number of exponent bits. Figure 10 depicts the exponent masking process. A bitmask is applied to remove the unrepresentable mantissa bits. The resulting method is an imperfect approximation of reducing floating-point bit count. However, it has the advantage of being fast, causing at most a doubling of the time per training step.

We apply reduced precision operations only on the matrix multiplication computations of the model, excluding the attention computation, which uses the Flash Attention v2 kernel. Following existing FP8 libraries such as TransformerEngine (NVIDIA, 2023), we separate the effects of reducing the computation's precision from reducing the data's precision in storage. As a result, the activations and model weights are kept at their original precision while the inputs and outputs of matrix multiplication are dynamically masked to emulate reduced precision computation with a high-precision accumulator. All states are kept in their original precision, and all operations other than matrix multiplication are performed in their original precision. In Figure 4, we include a diagram indicating the precision of the states and computations in a Llama decoder block.

4. Results

4.1. MS-AMP experiments

We first analyze the effect of real-world FP8 training by applying MS-AMP (Peng et al., 2023) v0.4.0 level O1. All experiments are run on an H100 node to ensure hardware availability of FP8 with 8 GPUs. Two sets of experiments are performed. In the first, shown in Figure 5, we train a Llama 120M model on a subsample of the FineWeb Edu (Penedo et al., 2024) dataset. In the second, shown in Figure 6, we train a GPT-2 124M model from the nanoGPT (Karpathy, 2022) codebase using the OpenWebText (Gokaslan & Cohen, 2019) dataset.

From the two experiments, we come to the following conclusions. In Figure 5, where we train a Llama model on FineWeb Edu, a well-known "clean" dataset, we do not observe a difference between the convergence rates of MS-AMP O1 models and BF16 reference models. However, in Figure 6, we see a divergence between the MS-AMP O1 results and the BF16 baseline that does not close, even after applying six times the training iterations for the MS-AMP model at 120K training steps. The differences are much greater when the LM head is included in the quantization, indicating that parts of the model are highly sensitive to the quantization, requiring further investigation into model architecture choices.

These results indicate that the FP8 training scheme in MS-AMP may not converge to the same loss as BF16 training or requires more training steps, depending on factors such as data quality. This strengthens our case that FP8 training may introduce hidden instabilities that are not evident until stress tested against circumstances that were not considered in the original works proposing them.

LLM pretraining datasets in production environments, especially for customized use-cases, are usually much "dirtier" than those of popular open-source datasets. For example, extensive data filtering may not be an option for low-resource languages or for subfields such as medical or legal data. Also, for newer domains such as video or robotic motion, well-established metrics of data quality do not yet exist. Therefore, the finding that a method of FP8 training may work well only on "clean" data would be a consequential one for LLM training in production.

4.2. Bit reduction experiments

We first attempt to identify the points where training instability becomes visible. The emulated reduced-precision representations are denoted using the number of exponent and explicit mantissa bits used. For example, standard BF16 is referred to as E8M7, while a floating-point number with its exponent clamped to seven bits and mantissa clamped to six bits is referred to as E7M6.

We find that removing even a single exponent bit prevents training altogether, resulting in the model failing to progress with any learning using E7M7, confirming previous findings (Henry et al., 2019) that neural network training is more sensitive to exponent bits than mantissa bits. To analyze the cause, we conduct an ablation on the clamping mechanism by either removing only the inner or outer exponent range, as depicted in Figure 10 in the Appendix. We find that models with only their inner exponent ranges clamped train normally while models with only their outer exponent ranges clamped do not, indicating that the inability to rep-

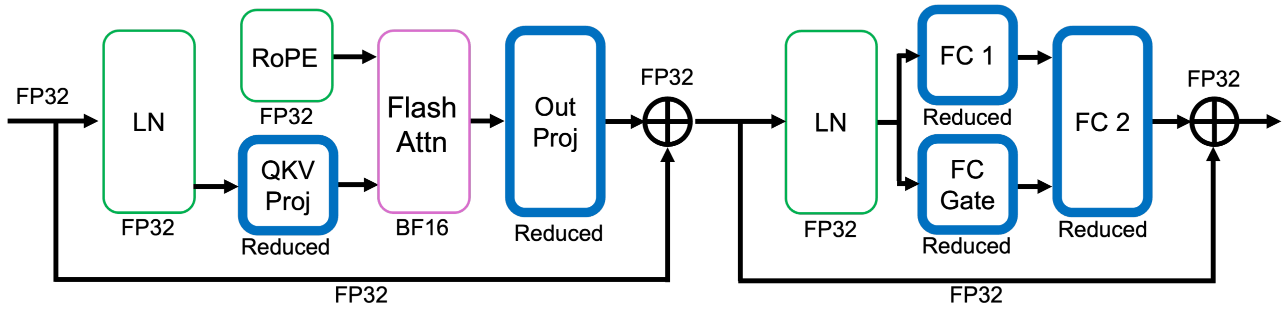

Figure 4. Diagram showing the precisions used in a Llama decoder block (best seen in color). The activations in the path of the residual connection are kept in FP32, as are the model weights and embeddings. The LayerNorm and RoPE (Su et al., 2024) layers use FP32 internally for their computations. The Flash Attention kernel uses BF16 with no reduction in precision. All other layers use reduced-precision matrix multiplication that emulates low-precision computation with a high-precision accumulator.

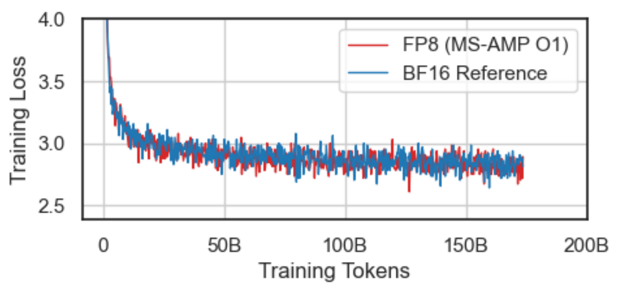

Figure 5. Comparison between MS-AMP FP8 (O1) and BF16 training on a subsample of the FineWeb Edu dataset using a Llama 120M model.

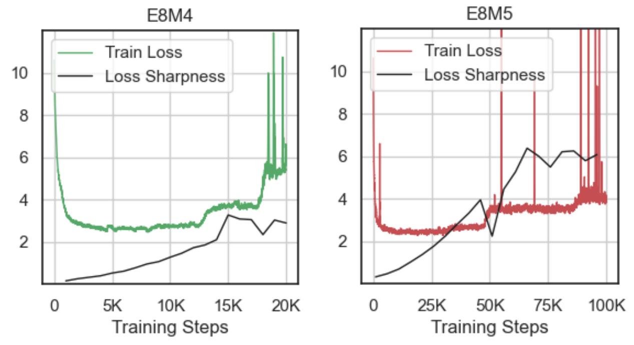

Figure 7. TinyLlama 120M models trained until loss divergence. E8M4 and E8M5 models are trained for 20K and 100K steps, respectively. The dotted black line in each figure indicates the loss landscape sharpness of the model.

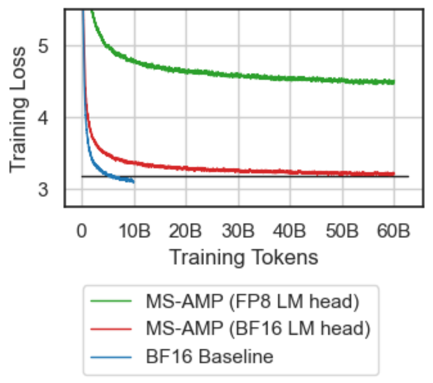

Figure 6. Training losses for GPT-2 training compared using exponential moving averages to better visualize the general trends. The blue curve indicates the training loss for the baseline BF16 training, while the red curve indicates MS-AMP level O1 training with the LM head excluded from FP8 quantization. The green curve shows the training loss for when the LM head included in the FP8 quantization.

resent large values is the cause of failure for E7M7. We therefore investigate the effects of removing mantissa bits for the remainder of our experiments.

4.3. Loss landscape sharpness

To further uncover the relationship between bit width and training robustness, we use Equation 2 to quantify the degree of training instability increase by measuring the loss landscape instability of Llama models. In Figure 7, we show Llama 120M models trained until their training losses diverge, as well as plotting the loss landscape sharpness values of the models in E8M4 and E8M5. Training required approximately 2 hours per 10,000 training steps on a node with 8 A100 GPUs.

Although the points of divergence are different for each model, we can see a general trend of increasing sharpness until the model diverges sharply, after which it cannot revert to its original training trajectory. This pattern is visible despite the large differences in training steps.

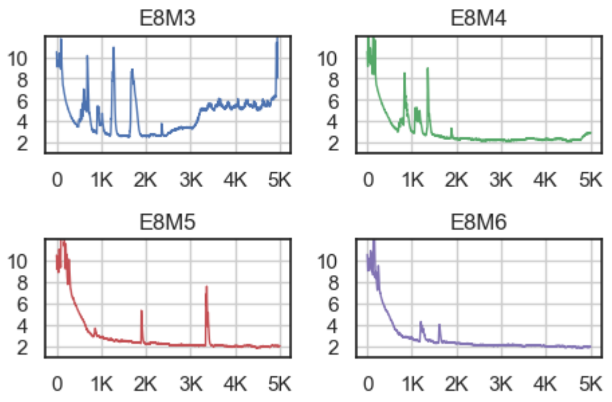

Figure 8. Llama 7B model training loss curves for different mantissa bits. The x-axis shows training steps, while the y-axis shows the training loss.

Steps	E8M3	E8M4	E8M5	E8M6	E8M7
1K	0.209	0.205	0.191	0.191	0.192
2K	0.488	0.363	0.265	0.221	0.200
3K	1.306	0.734	0.352	0.229	0.200
4K	2.006	1.125	0.475	0.237	0.207
5K	1.927	1.439	0.628	0.248	0.215

Table 1. Loss landscape sharpness values at $\epsilon = 5e{-}4$ for Llama v2 7B models trained with TinyLlama for 5,000 steps in Figure 8. Training used a global batch size of 512 and a sequence length of 4096.

To verify that similar behavior is reproducible in larger models, we compare the training losses of Llama 7B models trained for 5,000 steps in Figure 8 and show the measured sharpness values for $\epsilon = 5e{-}4$ in Table 1. Results for other ϵ values are included in Table 2 of the Appendix and show a similar pattern. We apply early stopping at 5,000 training steps because training a Llama 7B model for 5K steps requires approximately one week on a single node with 8 A100 GPUs. These experiments show that loss divergence is visible in the E8M3 and E8M4 models, while it has yet to emerge in the E8M5 model. However, from Table 1, we can see that the loss landscape sharpness continues to increase for the E8M5 model, even though no signs of instability are yet visible in the training loss.

The E8M3 and E8M4 models show much higher sharpness values, and both diverge early in training. In contrast, there is only a gradual increase in the loss-landscape sharpness for the E8M7 runs. Figures 7 and 8 show that models gradually increase in sharpness until a threshold level is reached. However, the exact threshold may differ depending on the configurations. These results suggest that models with fewer mantissa bits enter regions of ever greater instability during

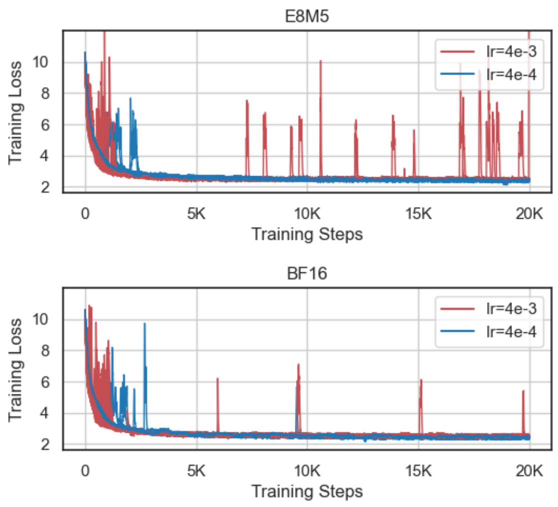

Figure 9. Comparison between Llama 120M models trained using E8M5 masked training (left) and standard BF16 training (right) for $lr = 4e{-}4$ (the default learning rate) and $lr = 4e{-}3$. Using 18 random seeds per configuration, the E8M5 runs show more frequent loss spikes, especially at the higher learning rate, indicating greater training instability.

training, even when these instabilities are not visible in the loss curve. We believe that, in the future, such analysis of loss landscape sharpness can be used to identify when the model is at risk of training loss divergence.

To verify that our proposed loss landscape sharpness metric tracks the training instability when it decreases as well as when it increases, we show the loss landscape sharpness trend for the E8M3 model when training is reverted to BF16 in Table 3 of the Appendix. Supporting our claim that the loss landscape sharpness is an indicator of training instability, the sharpness value continues to decrease until it nears the level of BF16 training as shown in Table 1.

4.4. Robustness to learning rate changes

We further attempt to identify hidden instability in E8M5, which did not diverge during the initial training stages in Figure 7. Inspired by Wortsman et al. (2024), we analyze the robustness of Llama 120M models to changes in the learning rate by comparing training at BF16 with that for E8M5. As seen in Figure 9, the E8M5 training runs have more frequent loss spikes during training, especially when the learning rate is increased to $4e{-}3$. Although no cases of loss divergence were found, we believe that the higher frequency of loss spikes indicates greater sharpness of the loss landscape, supporting our claim that training is more unstable for E8M5 even before loss divergence occurs.

5. Discussion

This work proposes quantitative evaluations and analyses of training instabilities when reducing floating-point precision. Our experiments have shown that MS-AMP, an existing open-source implementation of FP8 training, does not offer sufficient robustness to allow its cost-effective use. By emulating lower-precision mantissa bits in matrix multiplication, we further analyze the increasing instabilities arising from lower precision in LLM training.

Again, we clarify that we are not arguing that FP8 training is not viable. Indeed, we discuss the unique characteristics of DeepSeek V3 FP8 training in Section 2.1 and reproduce stable training using FP8 using the MS-AMP library in Section 4.1. The issue is that FP8 training causes a narrowing of the hyperparameter space where LLM training can occur stably and with equivalent performance as mixed-precision BF16 training. Because of this narrower hyperparameter space, more resources must be expended on identifying and honing techniques for preventing training collapse. Worse, there is simply no way of knowing if the FP8 training is performing competitively as BF16 without implementing a BF16 training run for comparison, which would negate the purpose of using FP8 for training in the first place. Even if FP8 training is practical for carefully selected hyperparameters under specific conditions, we assert that the costs of finding such conditions and the risks involved in training a less stable model outweigh the benefits of using FP8 for accelerated computation. Because of this risk, we propose methods to evaluate the stability and robustness of reduced-precision training, which is vital for FP8 or other reduced-precision training schemes to be viable for real-world LLM training.

From our experiments, several methods naturally suggest themselves as possible stabilization techniques. First, the initial stages of training could be conducted in higher precision, similar to how smaller batch sizes may be used during the initial stages of training as in Keskar et al. (2017). Increasing the precision when the loss landscape becomes too sharp may also provide a tradeoff between training speed and stability. Second, the more sensitive layers may be kept at high precision, while only the less sensitive layers are computed with reduced precision. For example, in Figure 11, we found that removing masking from the LM head of a Llama model was sufficient to enable E7M7 training to progress, although the resulting model was unstable. For the instability in GPT models shown in Figure 1, we found that increasing the precision of the first two decoder blocks to TF32 was sufficient to prevent loss divergence. However, as such compensatory techniques depend on the model architecture, training data, and other aspects of the training environment, we leave their investigation to future work.

6. Limitations

A limitation of this work is that it focuses on the initial stages of pre-training when many instabilities are known to arise only later in training (Fishman et al., 2024; Bekman, 2023). For example, Wortsman et al. (2024) show that the logits of the outputs diverge from zero only at the later stages of training, and Fishman et al. (2024) show that correlations between the SwiGLU layers only begin to increase after 200B tokens. To this, we argue that our studies likely underestimate the instabilities that FP8 or other reduced precision training schemes will face, further strengthening our argument that currently available FP8 training methods are too unstable to be profitably utilized for LLM training in their current form and that much more extensive evaluations are required to stress-test them.

Second, despite finding that the exponent bits are of greater importance to LLM training than mantissa bits, we were unable to experiment by increasing the number of exponent bits. This was because matrix multiplication in FP64 is over an order of magnitude slower than BF16 on A100 and H100 GPUs when using tensor cores. Experiments using representations such as E11M4, created by removing 48 mantissa bits from FP64, may be illuminating, but we found it impractical to train models with a greater number of exponent bits due to hardware limitations.

Finally, our experiments are limited in that they only the training loss is used as an evaluation metric instead of real-world natural language tasks such as MMLU (Hendrycks et al., 2021) scores. However, while lower perplexity does not guarantee superior performance on downstream tasks, we believe that training loss divergence is sufficient to indicate training failure.

7. Conclusion

We demonstrate that the training stability of LLMs decreases incrementally with the reduction of floating-point bit widths used for training models of up to 7B parameters. Using our proposed loss landscape sharpness metric, we measure the gradual narrowing of the hyperparameter space where stable training is possible.

In future works proposing reduced-precision training, we hope that our work inspires analyses of the robustness of newly proposed training methods not only on the well trodden paths with well-established guidelines, but also on less explored conditions where many of the initial assumptions may not hold. The massive power and capital consumption of LLM training means that much is at stake on this issue.

References

Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016.

Bekman, S. Machine learning: Llm/vlm training and engineering by stas bekman, 2023. URL https://stasosphere.com/machine-learning/.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Dao, T. FlashAttention-2: Faster attention with better parallelism and work partitioning, 2023.

DeepSeek-AI. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., Jenatton, R., Beyer, L., Tschannen, M., Arnab, A., Wang, X., Riquelme Ruiz, C., Minderer, M., Puigcerver, J., Evci, U., Kumar, M., Steenkiste, S. V., Elsayed, G. F., Mahendran, A., Yu, F., Oliver, A., Huot, F., Bastings, J., Collier, M., Gritsenko, A. A., Birodkar, V., Vasconcelos, C. N., Tay, Y., Mensink, T., Kolesnikov, A., Pavetic, F., Tran, D., Kipf, T., Lucic, M., Zhai, X., Keysers, D., Harmsen, J. J., and Houlsby, N. Scaling vision transformers to 22 billion parameters. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 7480–7512. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/dehghani23a.html.

Fishman, M., Chmiel, B., Banner, R., and Soudry, D. Scaling fp8 training to trillion-token llms, 2024. URL https://arxiv.org/abs/2409.12517.

Gokaslan, A. and Cohen, V. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Henry, G., Tang, P. T. P., and Heinecke, A. Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pp. 69–76, 2019. doi: 10.1109/ARITH.2019.00019.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, jun 2017. ISSN 0163-5964. doi: 10.1145/3140659.3080246. URL https://doi.org/10.1145/3140659.3080246.

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.

Karpathy, A. The simplest, fastest repository for training/finetuning medium-sized GPTs., 2022. URL https://github.com/karpathy/nanoGPT.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=H1oyRlYgg.

Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M., and Keutzer, K. Squeezellm: Dense-and-sparse quantization. *arXiv*, 2023.

Kroening, D. and Strichman, O. *Decision Procedures*. Springer, 2008.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In *Neural Information Processing Systems*, 2018.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M.-H., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Murthy, R., Stillerman, J., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. Starcoder: may the source be with you!, 2023.

Liu, D. C. and Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, August 1989.

Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., Mellempudi, N., Oberman, S., Shoeybi, M., Siu, M., and Wu, H. Fp8 formats for deep learning, 2022.

NVIDIA. Nvidia a100 tensor core gpu architecture, 2020. URL https://resources.nvidia.com/en-us-genomics-ep/ampere-architecture-white-paper.

NVIDIA. Nvidia h100 tensor core gpu architecture overview, 2022. URL https://resources.nvidia.com/en-us-tensor-core.

NVIDIA. TransformerEngine, 2023. URL https://github.com/NVIDIA/TransformerEngine.

OpenAI. Gpt-4 technical report, 2024.

Penedo, G., Kydlíček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Peng, H., Wu, K., Wei, Y., Zhao, G., Yang, Y., Liu, Z., Xiong, Y., Yang, Z., Ni, B., Hu, J., Li, R., Zhang, M., Li, C., Ning, J., Wang, R., Zhang, Z., Liu, S., Chau, J., Hu, H., and Cheng, P. Fp8-lm: Training fp8 large language models, 2023.

Shazeer, N. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Shen, H., Mellempudi, N., He, X., Gao, Q., Wang, C., and Wang, M. Efficient post-training quantization with fp8 formats. In Gibbons, P., Pekhimenko, G., and Sa, C. D. (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 483–498, 2024.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.127063. URL https://www.sciencedirect.com/science/article/pii/S0925231223011864.

Sun, M., Chen, X., Kolter, J. Z., and Liu, Z. Massive activations in large language models. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. URL https://openreview.net/forum?id=1ayU4fMqme.

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V. V., Cui, X., Zhang, W., and Gopalakrishnan, K. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/65fc9fb4897a89789352e211ca2d398f-Paper.pdf.

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_files/paper/2020/file/

13b919438259814cd5be8cb45877d577-Paper.pdf.

Team, G. Gemini: A family of highly capable multimodal models, 2024a.

Team, G. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024b.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. Training deep neural networks with 8-bit floating point numbers. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/335d3d1cd7ef05ec77714a215134914c-Paper.pdf.

Wang, R., Gong, Y., Liu, X., Zhao, G., Yang, Z., Guo, B., Zha, Z., and Cheng, P. Optimizing large language model training using fp4 quantization, 2025. URL https://arxiv.org/abs/2501.17116.

Wortsman, M., Liu, P. J., Xiao, L., Everett, K. E., Alemi, A. A., Adlam, B., Co-Reyes, J. D., Gur, I., Kumar, A., Novak, R., Pennington, J., Sohl-Dickstein, J., Xu, K., Lee, J., Gilmer, J., and Kornblith, S. Small-scale proxies for large-scale transformer training instabilities. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=d8w0pmvXbZ.

Wulf, W. A. and McKee, S. A. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

Xi, H., Cai, H., Zhu, L., Lu, Y., Keutzer, K., Chen, J., and Han, S. Coat: Compressing optimizer states and activation for memory-efficient fp8 training, 2025. URL https://arxiv.org/abs/2410.19313.

Yoo, K. M., Han, J., In, S., Jeon, H., Jeong, J., Kang, J., Kim, H., Kim, K.-M., Kim, M., Kim, S., Kwak, D., Kwak, H., Kwon, S. J., Lee, B., Lee, D., Lee, G., Lee, J., Park, B., Shin, S., Yu, J., Baek, S., Byeon, S., Cho, E., Choe, D., Han, J., Jin, Y., Jun, H., Jung, J., Kim, C., Kim, J., Kim, J., Lee, D., Park, D., Sohn, J. M., Han, S., Heo, J., Hong, S., Jeon, M., Jung, H., Jung, J., Jung, W., Kim, C., Kim, H., Kim, J., Kim, M. Y., Lee, S., Park, J., Shin, J., Yang, S., Yoon, J., Lee, H., Bae, S., Cha, J., Gylleus, K., Ham, D., Hong, M., Hong, Y., Hong, Y., Jang, D., Jeon, H., Jeon, Y., Jeong, Y., Ji, M., Jin, Y., Jo, C., Joo, S., Jung, S., Kim, A. J., Kim, B. H., Kim, H., Kim, J., Kim, M., Kim, M., Kim, S., Kim, Y., Kim, Y., Kim, Y., Ko, D., Lee, D., Lee, H. Y., Lee, J., Lee, J., Lee, J., Lee, J., Lee, M. Y., Lee, Y., Min, T., Min, Y., Moon, K., Oh, H., Park, J., Park, K., Park, Y., Seo, H., Seo, S., Sim, M., Son, G., Yeo, M., Yeom, K. H., Yoo, W., You, M., Ahn, D., Ahn, H., Ahn, J., Ahn, S., An, C., An, H., An, J., An, S.-M., Byun, B., Byun, E., Cha, J., Chang, M., Chang, S., Cho, H., Cho, Y., Choi, D., Choi, D., Choi, H., Choi, M., Choi, S., Choi, S., Choi, W., Chun, S., Go, D. Y., Ham, C., Han, D., Han, J., Hong, M., Hong, S. B., Hwang, D.-H., Hwang, S., Im, J., Jang, H. J., Jang, J., Jang, J., Jang, S., Jang, S., Jeon, J., Jeong, D., Jeong, J., Jeong, K., Jeong, M., Jin, S., Jo, H., Jo, H., Jo, M., Jung, C., Jung, H., Jung, J., Jung, J. H., Jung, K., Jung, S., Ka, S., Kang, D., Kang, S., Kil, T., Kim, A., Kim, B., Kim, B., Kim, D., Kim, D.-G., Kim, D., Kim, D., Kim, E., Kim, E., Kim, G., Kim, G. R., Kim, H., Kim, H., Kim, I., Kim, J., Kim, J., Kim, J., Kim, M., Kim, M., Kim, P. H., Kim, S., Kim, S., Kim, S., Kim, S., Kim, S., Kim, S., Kim, S., Kim, T., Kim, W., Kim, Y., Kim, Y. J., Kim, Y., Kwon, B., Kwon, O., Kwon, Y.-H., Lee, A., Lee, B., Lee, C., Lee, D., Lee, D., Lee, H.-R., Lee, H., Lee, H., Lee, H., Lee, I., Lee, J., Lee, J., Lee, J., Lee, J., Lee, J., Lee, J., Lee, J. H., Lee, J., Lee, J., Lee, S. Y., Lee, S., Lee, S., Lee, S., Lee, W., Lee, Z. H., Lim, J. K., Lim, K., Lim, T., Na, N., Nam, J., Nam, K.-M., Noh, Y., Oh, B., Oh, J.-S., Oh, S., Oh, Y., Park, B., Park, C., Park, D., Park, H., Park, H. T., Park, H., Park, J., Park, J., Park, J., Park, J., Park, M., Park, S. H., Park, S., Park, S., Park, T., Park, W., Ryu, H., Ryu, J., Ryu, N., Seo, S., Seo, S. M., Shim, Y., Shin, K., Shin, W., Sim, H., Sim, W., Soh, H., Son, B., Son, H., Son, S., Song, C.-Y., Song, C., Song, K. Y., Song, M., Song, S., Wang, J., Yeo, Y., Yi, M. Y., Yim, M. B., Yoo, T., Yoo, Y., Yoon, S., Yoon, Y. J., Yu, H., Yu, U. S., Zuo, X., Bae, J., Bae, J., Cho, H., Cho, S., Cho, Y., Choi, T., Choi, Y., Chung, J., Han, Z., Heo, B., Hong, E., Hwang, T., Im, S., Jegal, S., Jeon, S., Jeong, Y., Jeong, Y., Jiang, C., Jiang, J., Jin, J., Jo, A., Jo, Y., Jung, H., Jung, J., Kang, S., Kim, D. H., Kim, G., Kim, H., Kim, H., Kim, H., Kim, H., Kim, H.-A., Kim, J., Kim, J.-H., Kim, J., Kim, J., Kim, J. Y., Kim, R. Y., Kim, S., Kim, S., Kim, S., Kim, S., Kim, S., Kim, T., Ko, N., Koo, B., Kwak, H., Kwon, H., Kwon, Y., Lee, B., Lee, B. W., Lee, D., Lee, E., Lee, E., Lee, H. G., Lee, H., Lee, H., Lee, J., Lee, J., Lee, J., Lee, J., Lee, J., Lee, M., Lee, N., Lee, S., Lee, S. Y., Lee, S., Lee, S. J., Lee, S., Lee, Y., Lee, Y., Lee, Y., Lee, Y., Li, S., Liu, T., Moon, S.-E., Moon, T., Nihlenramstroem, M.-L., Oh, W., Oh, Y., Park, H., Park, H., Park, J., Park, N., Park, S., Ryu, J., Ryu, M., Ryu, S., Seo, A., Seo, H., Seo, K., Shin, J., Shin, S., Sin, H., Wang, J., Wang, L., Xiang, N., Xiao, L., Xu, J., Yi, S., Yoo, H., Yoo, H., Yoo, H., Yu, L., Yu, Y., Yuan, W., Zeng, B., Zhou, Q., Cho, K., Ha, J.-W., Park, J., Hwang, J., Kwon, H. J., Kwon, S., Lee, J., Lee, S., Lim, S., Noh, H., Choi, S., Lee, S.-W., Lim, J. H., and Sung, N. Hyperclova x technical report, 2024.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model, 2024.

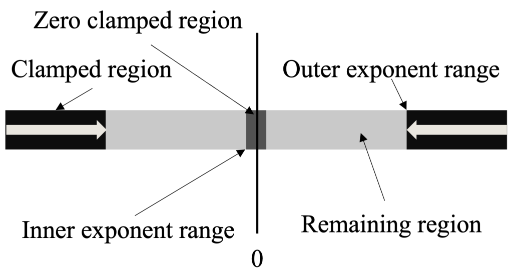

Figure 10. Exponent masking by clamping values not expressible with the given number of exponent bits.

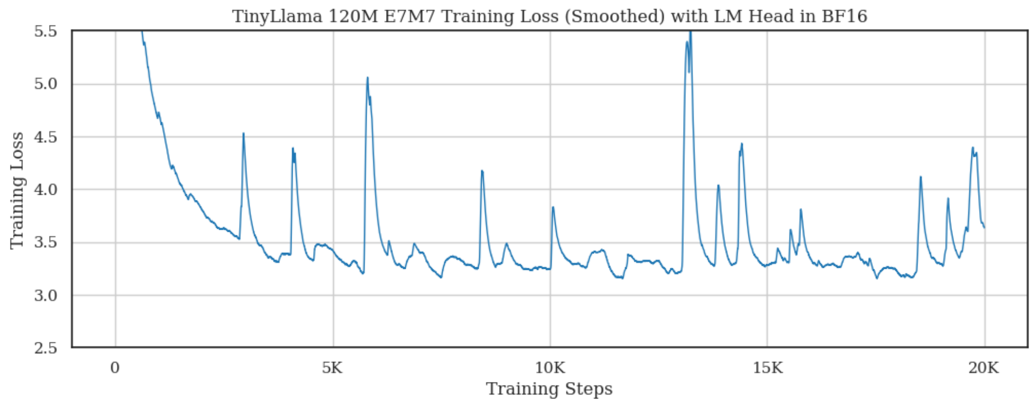

Figure 11. The training loss of a TinyLlama 120M model with clipped exponent at E7M7, excluding the LM head. The training loss is smoothed using exponential moving averages for better visualization. The results show that the exponent clipped models remain unstable when models in Figure 9 have stabilized after the same number of training steps.

```python
def backward(inputs, weight, output_gradient):
    masked_inputs = reduce_precision(inputs)
    masked_weight = reduce_precision(weight)
    masked_output_gradient = reduce_precision(output_gradient)
    inputs_gradient = F.linear(masked_inputs, masked_weight.T)
    weight_gradient = F.linear(masked_output_gradient.T, masked_weight.T)
    masked_inputs_gradient = reduce_precision(inputs_gradient)
    masked_weight_gradient = reduce_precision(weight_gradient)
    return masked_inputs_gradient, masked_weight_gradient
```

Figure 12. PyTorch-like pseudocode for the reduced-precision backward pass.

Table 2. Robustness of the sharpness metric to ϵ. We have found empirically that the loss landscape sharpness metric is robust to the choice of ϵ. Below, we include a table with sharpness values for a wide range of ϵ values for a Llama 7B model trained for 5,000 steps. We use a different checkpoint from the one in Table 1 of the paper to demonstrate reproducibility.

ϵ	Precision	1K	2K	3K	4K	5K
5.00E-05	E8M3	0.02	0.06	0.18	0.19	0.17
	E8M4	0.02	0.04	0.08	0.13	0.17
	E8M5	0.02	0.03	0.04	0.05	0.07
	E8M6	0.02	0.02	0.02	0.02	0.03
	E8M7	0.02	0.02	0.02	0.02	0.02
1.00E-04	E8M3	0.04	0.11	0.36	0.38	0.33
	E8M4	0.04	0.08	0.16	0.26	0.34
	E8M5	0.04	0.05	0.07	0.10	0.14
	E8M6	0.04	0.04	0.04	0.05	0.05
	E8M7	0.03	0.04	0.04	0.04	0.04
5.00E-04	E8M3	0.19	0.51	1.70	1.80	1.49
	E8M4	0.18	0.37	0.74	1.22	1.54
	E8M5	0.18	0.25	0.34	0.48	0.64
	E8M6	0.18	0.21	0.21	0.23	0.25
	E8M7	0.16	0.18	0.17	0.19	0.19
1.00E-03	E8M3	0.38	0.98	3.31	3.59	2.81
	E8M4	0.36	0.71	1.42	2.32	2.86
	E8M5	0.34	0.49	0.66	0.92	1.21
	E8M6	0.34	0.41	0.40	0.44	0.48
	E8M7	0.30	0.35	0.34	0.37	0.38
5.00E-03	E8M3	1.73	4.47	13.64	11.64	9.58
	E8M4	1.61	3.26	6.43	9.87	11.39
	E8M5	1.55	2.25	2.92	4.10	5.28
	E8M6	1.55	1.87	1.85	2.03	2.21
	E8M7	1.36	1.60	1.53	1.71	1.73

Table 3. We show the loss landscape sharpness of a Llama 7B model initially trained with E8M3 precision for 6,000 training steps that was then trained with standard BF16. We can see that the sharpness indicator decreases in value with more training on BF16, indicating that it is capturing the increased stability of training that comes with BF16 over E8M3.

Train Step	Sharpness
7K	1.35
8K	1.14
9K	0.98
10K	0.90
11K	0.87
12K	0.77
13K	0.71
14K	0.63
15K	0.60
16K	0.59
17K	0.57
18K	0.50
19K	0.48
20K	0.48
21K	0.46
22K	0.44
23K	0.40
24K	0.40
25K	0.37
26K	0.34
27K	0.34
28K	0.34
29K	0.33
30K	0.35
31K	0.32
32K	0.32
33K	0.30
34K	0.29
35K	0.29
36K	0.28
37K	0.27
38K	0.26
39K	0.27
40K	0.27
41K	0.25
42K	0.23
43K	0.23
44K	0.24