A Fistful of Errors

A Discrete Formulation of Neural Network Behavior in Floating Point Representation

Weyl AI Research

January 2026

Abstract

Deep learning theory analyzes neural networks as continuous functions optimized over $\mathbb{R}^n$. This abstraction, while mathematically convenient, obscures the actual computation. Every operation occurs on a discrete, non-uniform lattice determined by floating-point representation. We argue this is not an approximation to be corrected but the actual object of study. Different precision stacks define different optimization problems with different dynamics, not nested approximations to a "true" real-valued surface. We model practical mixed-precision pipelines and show that even when lower-precision values embed in higher-precision formats, the realized loss and training dynamics differ. Continuous analysis is valid as perturbation theory when rounding events are infrequent. When they are not, the lattice dominates.

Scope of claims. *Proven/constructed:* Piecewise-constant cell structure of $\mathcal{L}_\Lambda$; existence of non-monotonicity across non-nested formats (fp16 vs bf16). *Observed:* Precision-swap irreversibility; SDPA collapse with residual recovery; non-monotonic SNR across formats. *Conjectured:* Task-dependent precision optima as a general phenomenon.

1 The Standard Framework

Assumption 1 (Universal). *Neural network training minimizes a loss function $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ via gradient descent on a continuous manifold.*

This assumption underlies nearly all theoretical work: convergence proofs [1], generalization bounds [2], loss landscape analysis [3], implicit bias theorems [4]. It is mathematically convenient but physically unrealized.

No computer has ever performed a computation in $\mathbb{R}$.

2 The Discrete Frame

Floating-point arithmetic operates on a discrete set. IEEE 754 binary32 [5] has approximately 4.3×10^9 distinct bit patterns. For normalized numbers, spacing scales with $|x|$ (density $\propto 1/|x|$); in the subnormal regime, spacing is uniform.

A neural network forward pass is a composition of thousands of such operations. The function actually computed is:

$$\tilde{f} : \Lambda^n \to \Lambda$$

where Λ is the floating-point lattice. This is not $f : \mathbb{R}^n \to \mathbb{R}$ plus noise. It is a *different function* defined on a *different domain*.

2.1 Precision Stacks

Real training pipelines use mixed precision [6]. We define a **precision stack**:

$$\mathcal{P} = (\Lambda_w, \Lambda_a, \Lambda_g, \Gamma_{\text{acc}}, \Gamma_{\text{opt}}, \mathcal{R}, \mathcal{S})$$

for weights, activations, gradients, matmul accumulators, optimizer state, rounding policy $\mathcal{R}$, and scaling policy $\mathcal{S}$. Here $\mathcal{R}$ encodes rounding mode (RN-even, stochastic), saturation behavior, and FTZ/DAZ status. $\mathcal{S}$ encodes scaling granularity (per-tensor/per-channel/per-block) and group size. Different $(\mathcal{R}, \mathcal{S})$ induce different cell geometries even at fixed bitwidth.

The realized model depends on where quantizers Q_Λ are inserted:

- **Pre-op:** Inputs to each primitive are in their respective Λ
- **Accumulation:** GEMMs accumulate in Γ_{acc} (often fp32), then round to the branch's Λ
- **Optimizer:** Updates computed in Γ_{opt} (often fp32), optionally projected to Λ_w

Even when $\Lambda_w \subset \Gamma_{\text{opt}}$, the dynamics differ across stacks because rounding events, saturation, and accumulator overflow occur at different points—defining different cell boundaries in parameter space.

Observation 1. *Different precision formats define different optimization problems.*

For IEEE binary formats with the same radix, lower-precision values embed in higher-precision: every fp8 value is exactly representable in fp32. However:

- **Non-nested pairs exist:** fp16 and bf16 are not nested—each contains values the other cannot represent (mantissa/exponent trade-off).
- **Dynamics differ even when nested:** Training in fp16 vs fp32 produces different rounding events, different gradient quantization, different overflow behavior [7].
- **The realized loss differs:** $\mathcal{L}_\Lambda(\theta)$ is piecewise constant over cells induced by rounding thresholds. Different Λ means different cell boundaries, different critical points, potentially different minima.

Format	**Exp bits**	**Mant bits**	**Nested in fp32?**
fp32	8	23	—
fp16	5	10	Yes
bf16	8	7	Yes
fp8 (e4m3)	4	3	Yes
fp8 (e5m2)	5	2	Yes
int8	–	8	No (uniform vs log)

fp16 $\not\subset$ bf16 and bf16 $\not\subset$ fp16

The key insight: even when $\Lambda_1 \subset \Lambda_2$, training on Λ_1 is not "Λ_2 with noise." It is optimization of a different piecewise-constant function with different dynamics.

3 Formal Core

Definition 1 (Quantizer)**.** *A quantizer for lattice $\Lambda \subset \mathbb{R}$ is a map $Q_\Lambda : \mathbb{R} \to \Lambda$ (typically round-to-nearest-even with saturation). Extend elementwise to tensors. For analysis on $\mathbb{R}^n$, define $\mathcal{L}^\sharp_\mathcal{P}(\theta) = \mathcal{L}_\mathcal{P}(Q_{\Lambda_w}(\theta))$.*

Definition 2 (Realized Loss)**.** *For a network with primitives $\{\phi_i\}$, the realized loss at precision stack $\mathcal{P}$ replaces each ϕ with appropriate $Q_\Lambda \circ \phi \circ Q_\Lambda$ insertions. The realized loss $\mathcal{L}_\mathcal{P} : \Lambda_w^n \to \Gamma_{acc}$ is a function on the weight lattice, not $\mathbb{R}^n$. All analysis assumes a deployment distribution $\mathcal{D}$ with bounded dynamic range.*

Proposition 1 (Piecewise Constant)**.** *With deterministic rounding and fixed activation/branch regimes, $\mathcal{L}^\sharp_\mathcal{P}(\theta)$ is locally constant on a* ***locally finite stratified partition*** *of $\mathbb{R}^n$. The boundaries are finite unions of C^1 hypersurfaces given by preimages of quantizer midpoints and primitive kinks/saturation thresholds. On any compact set $K \subset \mathbb{R}^n$, only finitely many strata intersect K. On any stratum, all realized quantized tensors are constant; hence $\mathcal{L}^\sharp_\mathcal{P}$ is constant.*

Proof sketch. (1) For fixed rounding decisions, each Q_Λ outputs a constant; compositions with smooth ϕ yield smooth functions quantized back to Λ. (2) Rounding decisions change when a preimage crosses a midpoint $x = c + \frac{1}{2}\mathrm{ULP}(c)$. With nonlinear ϕ, these preimages are generally curved hypersurfaces, not hyperplanes. (3) Finitely many thresholds per finite op graph induce a locally finite stratification (finite on any compact set). (4) On any stratum, realized tensors and thus $\mathcal{L}^\sharp_\mathcal{P}$ are constant. (5) Boundaries are measure zero. □

Note on ULP: ULP depends on the exponent; boundaries jump at powers of two. Many accelerators enable flush-to-zero (FTZ) and denormals-are-zero (DAZ), which change cell geometry and can induce large, structured plateaus near zero.

Corollary 1 (Stagnation)**.** *If $|\Delta\theta_i| < \frac{1}{2}ULP_i(\theta_i)$ for all i and no intermediate activation crosses a quantizer or nonlinearity threshold under the induced change, then $\mathcal{L}^\sharp_\mathcal{P}(\theta_{t+1}) = \mathcal{L}^\sharp_\mathcal{P}(\theta_t)$ despite $\|\nabla\mathcal{L}\| > 0$.*

This explains training plateaus: the optimizer computes updates that don't escape the current stratum. The loss surface is not smooth. It is a staircase. Gradient descent on a staircase either crosses a step or stands still.

3.1 Surrogate Gradients

If $\mathcal{L}^\sharp_\mathcal{P}$ is piecewise constant, its gradient is zero almost everywhere. How does training make progress?

Answer: Autodiff through casts either omits the rounding nonlinearity (treating casts as identity for backprop) or uses straight-through estimators (STE) [17]. Training follows a surrogate gradient of a smoothed proxy $\mathcal{L}_{\mathcal{P},\sigma}$ rather than $\nabla\mathcal{L}^\sharp_\mathcal{P}$.

This reconciles observed progress with a staircase objective:

- **STE:** Gradient flows through as if quantization were identity
- **Stochastic rounding:** Randomizes thresholds, optimizing an explicit smoothing of $\mathcal{L}^\sharp_\mathcal{P}$ by convolving with the rounding kernel
- **QAT:** Learns scales that reshape cell geometry during training

Continuous analysis is a good perturbative surrogate in the **low boundary-crossing regime**, where rounding decisions are stable over many steps and $\mathcal{L}$ correlates with $\mathcal{L}^\sharp_\mathcal{P}$ on visited strata.

Proposition 2 (Non-Monotonicity Existence)**.** *For non-nested pairs* (Λ_1, Λ_2) *such as (fp16, bf16), there exist tasks where*

$$\min_{\theta \in \Lambda_1^n} \mathcal{L}_{\Lambda_1}(\theta) < \min_{\theta \in \Lambda_2^n} \mathcal{L}_{\Lambda_2}(\theta)$$

and other tasks where the inequality reverses.

Proof (constructive). Consider $\mathcal{L}(x) = (x - \alpha)^2$ with α placed within half-ULP of a representable point in Λ_1 but not in Λ_2. The minimizer snaps to different grid points, yielding different losses. Swap α's position to reverse the inequality. $\square$

Proposition 3 (No Total Order)**.** *For any two precision stacks* $\mathcal{P}_a, \mathcal{P}_b$*—including cases where* $\Lambda_a \subset \Lambda_b$*—there exist tasks* T_1, T_2 *such that* $\mathcal{P}_a$ *achieves lower loss on* T_1 *and* $\mathcal{P}_b$ *achieves lower loss on* T_2*.*

Even nested formats can swap ordering because the dynamics (saturation, accumulation, cell boundaries) differ.

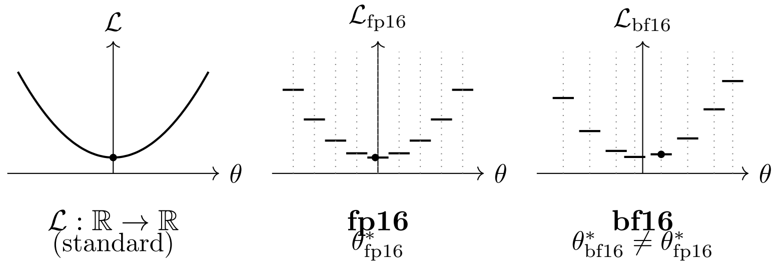
$\mathcal{L}$
θ
$\mathcal{L}_{\text{fp16}}$
θ
$\mathcal{L}_{\text{bf16}}$
θ
$\mathcal{L} : \mathbb{R} \to \mathbb{R}$
(standard)
fp16
θ^*_{fp16}
bf16
$\theta^*_{\text{bf16}} \neq \theta^*_{\text{fp16}}$

Figure 1: The continuous parabola (left) is the standard theoretical abstraction. In practice, only discrete surfaces exist, with different cell boundaries and different minima per precision. fp16 and bf16 are non-nested: each contains values the other cannot represent. Minima do not merge under $A \to B \to A$ precision casting.

4 The Fossil Record

Architectural innovations correlate with precision stack transitions:

Year	**Precision**	**Adaptation**
2012	fp32 GPU	ReLU (avoids exp overflow)
2015	fp32 GPU	BatchNorm (range control)
2016	fp32 GPU	ResNet (SNR preservation)
2017	fp32 GPU	$1/\sqrt{d_k}$, LayerNorm
2018+	bf16 TPU	Same constants reinforced
2019	fp16 GPU	GELU (smoother gradients)
2022	bf16 GPU	RMSNorm, SwiGLU
2024	fp8 GPU	Smooth-SwiGLU [16]

We do not claim these choices were made consciously for numerical reasons. We claim they survived because they correlate with selection pressures that stabilize ranges and signal-to-noise ratios on the precision stacks that dominated each era.

ReLU replaced sigmoid when fp32 made derivative discontinuity acceptable. SwiGLU emerged on bf16 but required modification for fp8 because it produces outlier activations exceeding representable range.

We are not doing architecture design. We are doing archaeology.

4.1 The Magic Numbers as Range Control

Reframe the Transformer constants as lattice-specific range controls:

- $1/\sqrt{d_k}$: Keeps logits in representable range after cast back to Λ_a. Reduces tendency toward near-one-hot attention under fixed learning rate. On $\mathbb{R}$, it normalizes variance. On the lattice, it maintains a healthy boundary-crossing rate.
- $d_k = 64$: One point in a precision-dependent band $\{32, 64, 96, 128\}$. Modern LLMs explore the full range.
- $d_{ff} \approx 4d_{\text{model}}$: Boundary-crossing rate stays healthy. Gated FFNs shift this to ≈ 3.5–$4.7\times$ effective expansion.
- 8 heads at 64 dims: Product of precision-safe head size and practical parallelism.

On a different lattice, different configurations survive.

5 Observations

We measured SNR (signal-to-noise ratio vs. fp32 reference) across 1,444 layer-precision pairs in three model families under dynamic activation quantization.

5.1 SDPA Collapse with Residual Recovery

Scaled dot-product attention shows dramatic SNR collapse at NVFP4:

Model	**SDPA min (dB)**	**Residual out (dB)**	**Recovery**
Qwen2.5-1.5B	−6.33	22.1	+28 dB
FLUX.1-schnell	1.17	28.4	+27 dB
BERT-base	8.88	31.2	+22 dB

Qwen shows *negative* SNR—noise exceeds signal. Yet the model produces coherent output. The residual stream acts as a carrier wave; when it adds to the degraded attention output, coherent detection recovers the signal.

SNR combination (linear domain): For residual r and attention a paths with quantization noises $\varepsilon_r, \varepsilon_a$:

$$\text{SNR}_{\text{combined}} = \frac{S}{N_r + N_a + 2\,\text{cov}(\varepsilon_r, \varepsilon_a)}$$

With uncorrelated noise (cov ≈ 0) and coherent signal addition, the combined SNR exceeds either branch. Convert to dB only after linear combination. This is standard coherent detection from RF engineering [10]. **Verify empirically:** report $\text{corr}(y_{\text{fp}}, y_{\text{qa}} - y_{\text{fp}})$ and $\text{corr}(y_{\text{fp}}, y_{\text{qr}} - y_{\text{fp}})$ to justify the uncorrelated noise assumption.

Definition (NVFP4): A 4-bit floating-point format deployed with block-wise microscaling (e.g., group size 32). Each block shares a scale factor, improving dynamic-range utilization relative to per-tensor scaling. Verify exponent/mantissa layout and group size against vendor documentation; our results reflect the full stack $(\Lambda, \mathcal{S})$ rather than bitwidth alone.

5.2 Non-Monotonicity

FLUX.1-schnell mean layer SNR:

- Float8: 6.2 dB
- NVFP4: 40.4 dB

Four bits outperform eight bits. This reversal is inconsistent with "lower precision = more noise." It is consistent with different lattices having different geometry—block-wise microscaling in NVFP4 (group size 32) provides better dynamic range management than per-tensor Float8 scaling for this architecture.

Control note: This comparison requires controlling for scaling granularity (per-tensor vs per-channel vs block), clipping policy, and calibration window. The "4 ¿ 8 bits" result reflects the full precision stack, not bit count alone.

6 The Four Consequences

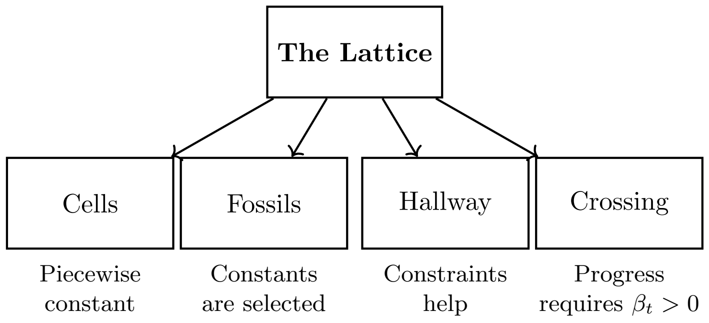

Figure 2: Four consequences of the Lattice Hypothesis: loss is piecewise constant on cells, architectural constants are precision-selected fossils, constraints improve signal-to-branching ratio, and progress requires boundary crossings.

7 Experimental Program

Four experiments to test the hypothesis:

7.1 Experiment 1: Learned Temperature

Replace $1/\sqrt{d_k}$ with learned per-head τ_h. Initialize to $\sqrt{d_k}$. Train identical architectures on fp8-e4m3, fp8-e5m2, fp16, bf16, fp32.

Prediction: τ_h converges to different values across precisions. Lower precision $\rightarrow$ higher τ (wider attention to compensate for quantization noise). Per-head variance is diagnostic.

7.2 Experiment 2: Head Dimension Sweep

Sweep $d_k \in \{32, 48, 64, 80, 96, 128\}$ at each precision, holding compute roughly fixed (adjust head count with d_{model} fixed). Log pre/post-scale logit std, softmax entropy, cast change rate.

Prediction: Optimal d_k shifts with precision. fp8 prefers smaller d_k; fp32 tolerates larger.

7.3 Experiment 3: FFN Width Sweep

Sweep expansion factor $\in \{2, 3, 4, 6, 8\}$ at each precision. Log boundary-crossing rate and ULP-normalized update histograms.

Prediction: Optimal width decreases with precision. The $4\times$ factor is fp32/bf16-specific.

7.4 Experiment 4: Architecture Search

Run differentiable architecture search with precision as environmental variable. Let search discover scaling, head dimension, FFN ratio, normalization.

Prediction: Different precisions yield different architectures. Search rediscovers 2017 constants on bf16, different constants on fp8.

8 Falsifiable Predictions

1. **Precision-order non-monotonicity.** There exist workloads where bf16 trains stably and fp16 diverges, and vice versa, at matched throughput and accumulator precision.
2. **Precision-stack hysteresis.** Swapping precision mid-training ($\mathcal{P}_A \to \mathcal{P}_B$ for k steps, then $\mathcal{P}_B \to \mathcal{P}_A$) does not recover the original loss trajectory.
3. **Boundary-crossing dynamics.** Training progress correlates with the boundary-crossing rate β_t: the fraction of parameters where $|\Delta\theta_i| > \frac{1}{2}\text{ULP}_i$. Plateaus coincide with near-zero crossing rates despite nonzero gradients.
4. **Stochastic rounding as exploration.** Enabling unbiased stochastic rounding [8, 9] increases boundary-crossing rates and improves optima in low-bit training.

8.1 Proposed Measurements

- **Weight boundary-cross rate β_t:** Fraction of parameters where $|\Delta\theta_i| > \frac{1}{2}\text{ULP}_i(\theta_i)$
- **Effective boundary-cross rate β_t^{eff}:** Counts rounding-decision flips anywhere in the forward graph, not just weights:

$$\beta_t^{\text{eff}} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}\{\text{cell\_id}_m(\theta_{t+1}) \neq \text{cell\_id}_m(\theta_t)\}$$

  where cell_id_m is the vector of rounding decisions for M probed tensors. This aligns the metric with the theory.
- **ULP-normalized step size:** Histogram of $|\Delta\theta_i|/\text{ULP}_i(\theta_i)$; progress requires mass above 0.5
- **Saturation rate:** Fraction of activations/weights hitting max finite value
- **Cell-ID churn:** Hamming distance of rounding decisions per step (use bit-pattern comparison via `view` as int32/64 for robustness to denormals)

9 Implications

For architecture search: Searches on fp32 find fp32-optimal architectures. Deploying on fp8 requires re-search.

For scaling laws: Current scaling laws are curves for specific precision stacks. Different lattices, different exponents.

For quantization: "Degradation" is not approximation error. It is a different objective function. Post-training quantization changes the problem; the solution may no longer be optimal.

For LoRA: The rank constraint is a hallway. It works not despite limiting expressivity but because it limits wrong moves. The optimal rank depends on the precision stack.

For hardware design: The architecture and precision are not separable. Accelerators designed for "Transformer inference" assume constants that may not transfer.

10 Likely Objections

"But continuous analysis works!" Agreed—when rounding events are infrequent and saturation rare. We formalize that regime as "low boundary-cross rate," where $\mathcal{L}$ is a good surrogate for $\mathcal{L}_{\mathcal{P}}$. The question is when this approximation breaks down.

"Nested sets mean approximation." Even with $\Lambda \subset \Gamma$, the realized objective and dynamics differ because rounding thresholds, accumulator precision, and saturation define different cell decompositions. Optimization traces are not perturbations of each other when boundary events dominate.

"This is just numerical analysis." We elevate it from error analysis to problem definition: the objective being optimized *changes* with the precision stack. The right question is not "how much error does low precision introduce?" but "which lattice has good solutions for this task?"

11 Instrumentation

Key diagnostics for any experiment:

- **Boundary-crossing rate β_t:** Fraction of weight updates $\geq \frac{1}{2}$ ULP
- **Cast change rate:** Fraction of values that change when cast between formats
- **Softmax entropy:** Low entropy = saturated attention
- **ULP-normalized updates:** Histogram of $|\Delta\theta|/\text{ULP}(\theta)$
- **Per-op dtype logging:** What precision at each operation?

If $\beta_t \to 0$, training stagnates regardless of gradient magnitude. If cast change rate is high at a specific layer, that layer is precision-sensitive.

11.1 Threats to Validity

1. **Backend specifics:** Operator dtypes, fused kernels, and reduction orders vary by framework/version; always log per-op dtypes.
2. **Accumulator exceptions:** Some kernels may not use fp32 accumulators; confirm via kernel dumps.

3. **Dynamic scaling:** Per-tensor/per-channel scales, loss scaling, and stochastic rounding alter cell geometry.
4. **Seed/optimizer effects:** Gauge selection varies; compare gauge-invariant vs gauge-dependent metrics explicitly.
5. **Compute matching:** When sweeping d_k or heads, keep FLOPs/activation-memory approximately fixed.

11.2 Reproducibility Checklist

Report: hardware, driver, framework commit, kernels enabled; seeds; tokenizer/data hash; batch size, LR schedule, gradient clipping; per-op dtypes (inputs, accumulators, outputs); stochastic rounding status; loss-scaling policy; checkpoint/optimizer state precision.

12 Conclusion: Not Even Long

"Not Even Wrong" describes claims that cannot be falsified. We propose "Not Even Long" for derivations that do not exist.

The fundamental constants of the Transformer—$d_k = 64$, $d_{ff} = 4d_{\text{model}}$, $1/\sqrt{d_k}$—are fossils. They were selected on fp32/bf16 stacks in 2017–2020. They propagated because they worked. They were rationalized post-hoc.

The research program is clear:

1. Measure boundary-crossing rates across precision stacks
2. Learn the "constants" and watch them vary
3. Run architecture search per precision
4. Build theory for hallway quality

The proof is not a theorem. The proof is an experiment.

Deep learning theory is perturbation theory around a continuous abstraction.
The lattice is the substrate. The rest is perturbation theory.

A Code

A.1 Core Utilities

```
import torch, math
import torch.nn as nn
import torch.nn.functional as F

def ulp_exact(x):
    """Exact ULP using nextafter. Handles subnormals."""
    inf = torch.tensor(float('inf'), dtype=x.dtype, device=x.device)
    ninf = torch.tensor(float('-inf'), dtype=x.dtype, device=x.device)
    xp = torch.nextafter(x, inf)
    xm = torch.nextafter(x, ninf)
    return torch.minimum((xp - x).abs(), (x - xm).abs())
    # Note: If FTZ/DAZ enabled, ULP is effectively infinite
    # below smallest normal. Check torch.finfo(x.dtype).tiny.

def bitwise_equal(a, b):
    """Elementwise bit-identity; robust to NaNs."""
    assert a.dtype == b.dtype and a.shape == b.shape
    if a.dtype == torch.float32:
        return a.view(torch.int32) == b.view(torch.int32)
    if a.dtype in (torch.float16, torch.bfloat16):
        return a.view(torch.int16) == b.view(torch.int16)
    raise NotImplementedError(f"Unsupported dtype {a.dtype}")

def boundary_crossing_rate(theta, delta):
    """Fraction of updates exceeding 0.5 ULP."""
    return (delta.abs() >= 0.5*ulp_exact(theta)).float().mean()

def cast_change_rate(x, from_dtype, to_dtype):
    """Fraction of values that change when cast (bitwise)."""
    x_from = x.to(from_dtype)
    x_to = x_from.to(to_dtype).to(from_dtype)
    return (~bitwise_equal(x_from, x_to)).float().mean()
```

A.2 Boundary-Crossing Logger

```
class CrossingLogger:
    """Track boundary-crossing rates per parameter."""
    def __init__(self, model):
        self.prev = {n: p.detach().clone()
                     for n,p in model.named_parameters()}

    def step(self, model):
        rates = {}
        for n, p in model.named_parameters():
            if not p.requires_grad:
                continue
            delta = p.detach() - self.prev[n]
            bcr = (delta.abs() >= 0.5*ulp_exact(self.prev[n]))
            rates[n] = bcr.float().mean().item()
            self.prev[n].copy_(p.detach())
        return rates
```

A.3 Per-Op Dtype Tracer

```
def attach_dtype_tracer(module: nn.Module):
    """Log input/output dtypes for all layers."""
    def hook(mod, inputs, output):
        def dtype_of(x):
            return getattr(x, 'dtype', None)
        ins = [dtype_of(t) for t in inputs
               if torch.is_tensor(t)]
        outs = ([dtype_of(output)]
                if torch.is_tensor(output)
                else [dtype_of(t) for t in output
                      if torch.is_tensor(t)])
        print(f"{mod.__class__.__name__}: "
              f"in={ins} out={outs}")
    for m in module.modules():
        if any(hasattr(m, a) for a in ['weight','bias']):
            m.register_forward_hook(hook)
```

A.4 Learned Temperature Attention

```
class LearnedTemperatureAttention(nn.Module):
    """Per-head learned temperature for Exp 1."""
    def __init__(self, d_model, n_heads):
        super().__init__()
        self.n_heads = n_heads
        self.d_k = d_model // n_heads
        init = math.log(self.d_k ** 0.5)
        self.log_tau = nn.Parameter(
            torch.full((n_heads,), init))

    def forward(self, Q, K, V):
        tau = self.log_tau.exp().view(1,-1,1,1)
        scores = Q @ K.transpose(-2,-1) / tau
        scores = scores - scores.max(-1,keepdim=True)[0]
        return F.softmax(scores, -1) @ V

    @property
    def tau(self):
        return self.log_tau.exp().detach().cpu().tolist()
```

References

[1] Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2), 223–311.

[2] Neyshabur, B., Bhojanapalli, S., McAllester, D., & Srebro, N. (2017). Exploring generalization in deep learning. *NeurIPS*.

[3] Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (2018). Visualizing the loss landscape of neural nets. *NeurIPS*.

[4] Gunasekar, S., Lee, J., Soudry, D., & Srebro, N. (2018). Characterizing implicit bias in terms of optimization geometry. *ICML*.

[5] IEEE 754-2019. Standard for floating-point arithmetic.

[6] Micikevicius, P., et al. (2018). Mixed precision training. *ICLR.*

[7] Wang, N., Choi, J., Brand, D., Chen, C. Y., & Gopalakrishnan, K. (2018). Training deep neural networks with 8-bit floating point numbers. *NeurIPS.*

[8] Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). Deep learning with limited numerical precision. *ICML.*

[9] Higham, N. J., & Pranesh, S. (2020). Stochastic rounding and its probabilistic backward error analysis. *SIAM J. Sci. Comput.*

[10] Proakis, J. G., & Salehi, M. (2007). *Digital Communications.* McGraw-Hill.

[11] Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). GPT3.int8(): 8-bit matrix multiplication for transformers at scale. *NeurIPS.*

[12] Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., & Han, S. (2023). SmoothQuant: Accurate and efficient post-training quantization for large language models. *ICML.*

[13] Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). GPTQ: Accurate post-training quantization for generative pre-trained transformers. *ICLR.*

[14] Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., & Han, S. (2024). AWQ: Activation-aware weight quantization for LLM compression and acceleration. *MLSys.*

[15] Nahshan, Y., et al. (2019). Loss aware post-training quantization. *arXiv:1911.07190.*

[16] DeepSeek-AI. (2024). DeepSeek-V3 technical report. *arXiv:2412.19437.*

[17] Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432.*