Scaling Laws for Precision

Tanishq Kumar*[1] Zachary Ankner*[3, 4] Benjamin F. Spector[2] Blake Bordelon[1] Niklas Muennighoff[2]
Mansheej Paul[4] Cengiz Pehlevan[1] Christopher Ré[2] Aditi Raghunathan[5]

[1]Harvard University
[2]Stanford University
[3]MIT
[4]Databricks
[5]Carnegie Mellon University

Presenter: Ava Zheer Wang, Joe Ye

Problem

Is more pretraining compute always better?

Standard intuition:
More data + more compute → better model

This Paper shows:
Not necessarily, if the model will be quantized after training

How should we trade off precision, parameters, and data?

Motivation: Why should we care about precision?

Precision affects both cost and quality

System reality

- Modern LLMs already use BF16, FP8, and quantization.
- Precision changes training cost, inference cost, and memory use

Gap in classical laws

- Standard scaling laws usually discuss model size and data
- Precision is often treated as an implementation detail

Paper's goal

- Build precision-aware scaling laws
- Cover both low-precision training and post-training quantization

Minimal Background: What is being quantized?

Notation

N = model size (parameters)

D = dataset size (tokens)

P_w, P_a, P_kv = training precision of weights, activations, KV cache

P_post = precision used for post-training quantization at inference

Two Settings

PTQ

Train normally → quantize at the end → serve

Low-precision training

Train directly with lower-precision weights / activations / KV

Why split weights, activations, and KV cache?

- Different workloads are bottlenecked by different things: pretraining is usually compute-bound, small-batch inference is often weight-bandwidth-bound, and long-context decoding is often KV-cache-bound

Standard Classical Scaling Law

$$L(N, D) = AN^{-\alpha} + BD^{-\beta} + E$$

More parameters → lower loss

More data → lower loss

Both have diminishing returns

Where is precision?

Experimental setup

- OLMo-style Transformer trained on Dolma v1.7
- Model sizes: N ∈ {30M, 60M, 110M, 220M}
- Data budgets: D ∈ {1.5B, 3B, 6B, 13B, 26B} tokens
- Training precisions swept from 3 to 16 bits across weights / activations / KV
- 465 pretraining runs, plus multiple post-train quantization settings

Why such small models?

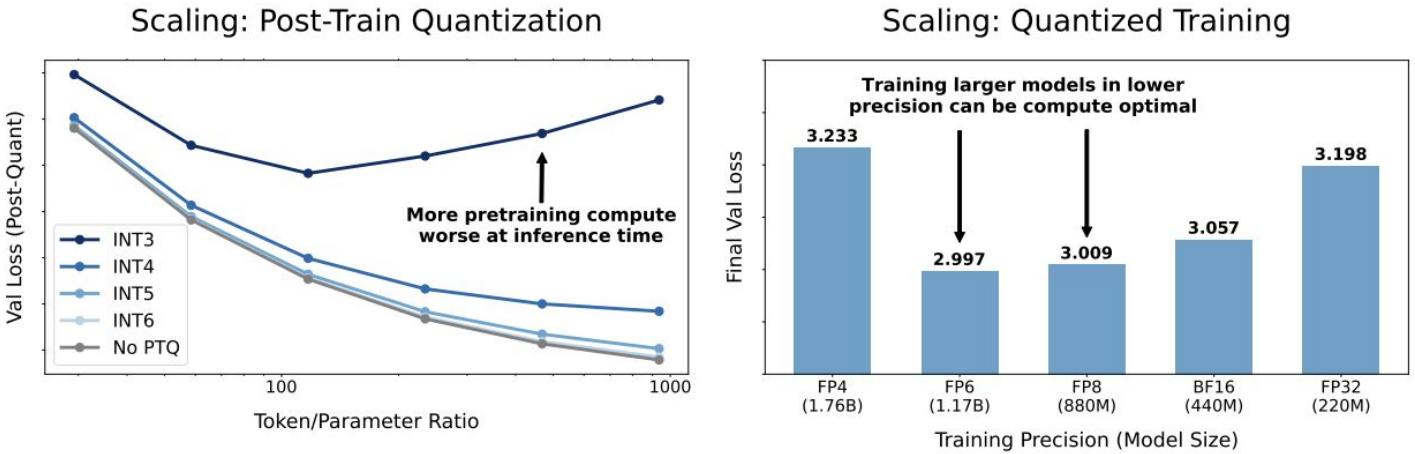
The goal is not frontier performance. The goal is to sweep many precision settings and push to very high token/parameter ratios.

Key regime

They intentionally study overtraining: D/N up to roughly 10^3 in the main experiments.

This is exactly where quantization sensitivity becomes interesting.

Big picture preview



Left: post-training quantization can get worse as the model sees more data

Right: at fixed compute, larger models in lower precision may be more efficient

First half of the talk: left panel Second half: right panel Final section: unify both

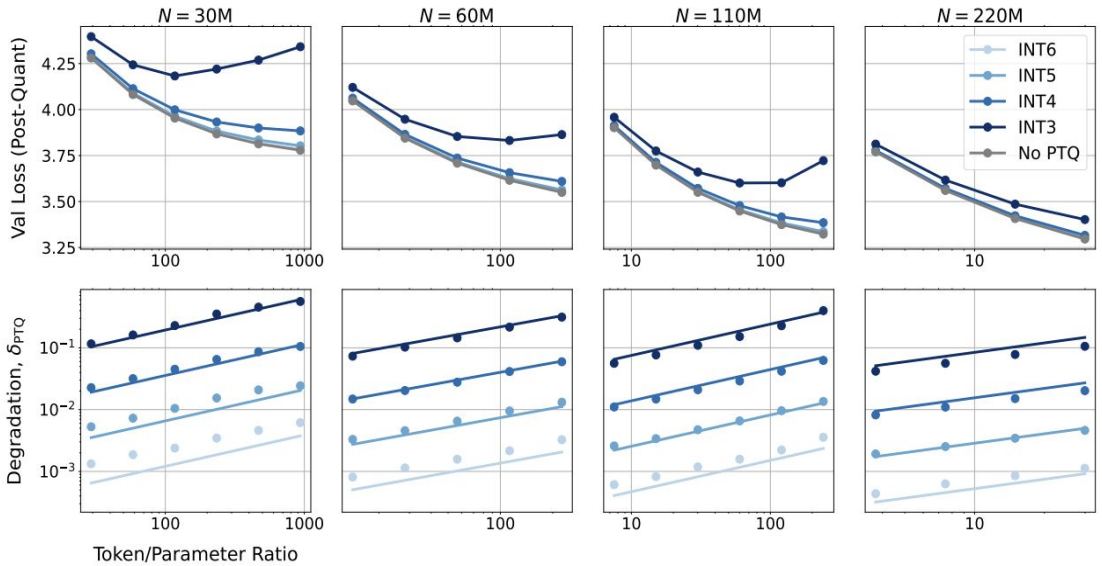
How does PTQ degradation scale?

Suppose a model is trained normally in BF16, then quantized only at the end, how does the induced degradation scale with model size N and data D?

BF16 pretraining → PTQ on weights → inference-time loss

The paper denotes this added loss by δ_PTQ

Overtrained models degrade more under PTQ



How to read the figure
Top row: loss after PTQ Bottom row: degradation relative to the unquantized model

Main message
degradation grows with data larger models are less sensitive at sufficiently large D/N, more data can actually worsen postquantized performance

Finding 1: PTQ degradation scales with data, size, and precision

$$\delta_{\text{PTQ}}(N, D, P_{\text{post}}) = C_T\left(\frac{D^{\gamma_D}}{N^{\gamma_N}}\right)e^{-P_{\text{post}}/\gamma_{\text{post}}}$$

More data D → More PTQ degradation

Larger model N → Less PTQ degradation

Lower post-training precision P_post → Much worse degradation

For models that will be post-train quantized, there can be a point where extra pretraining data becomes actively harmful at inference time.

First-half takeaway: PTQ is not just a deployment detail

If a model will be quantized after training, more pretraining data is not always better.

- PTQ degradation grows with token/parameter ratio
- Bigger models are more robust to PTQ
- Precision must enter scaling laws explicitly

So far, we treated precision as an inference-time choice.
Next: what if precision already changes the model during training?

From inference precision to training precision

Main question

Can precision during training be traded off against model size and data in a predictable way?

Transition

- Section 3: PTQ after BF16 training
- Section 4: quantized / low-precision training
- Goal: incorporate precision directly into scaling laws

Takeaway

Precision is not only an inference-time choice; it changes pretraining behavior too.

Core idea: lower precision reduces effective model capacity

A low-precision model with N parameters may behave like a higher-precision model with fewer than N “effective” parameters.

$$N_{\text{eff}} < N$$

Interpretation

- Extra bits improve usable capacity
- But returns saturate
- Precision and parameter count become partially interchangeable

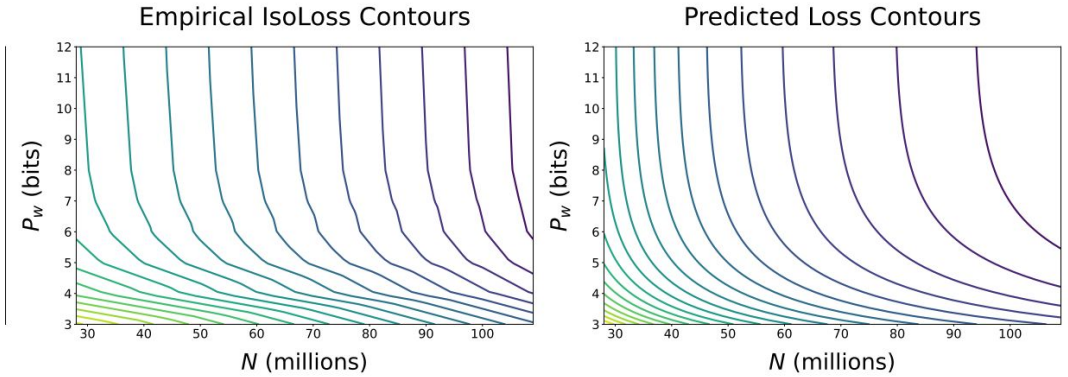
Why this matters

This converts a messy quantization problem into a scaling-law problem.

Step 1: weights-only quantization during training

Setting

- Quantize **weights** during training
- Keep activations and KV cache in high precision
- Study tradeoff between:
  - weight precision Pw
  - model size N



Observation

A larger model with lower Pw can match the loss of a smaller model with higher Pw.

A simple fit for weights-only precision

A simple fit for weights-only precision

$$N_{\text{eff}}(N, P_{\text{w}}) = N(1 - e^{-P_{\text{w}}/\gamma_{\text{w}}})$$

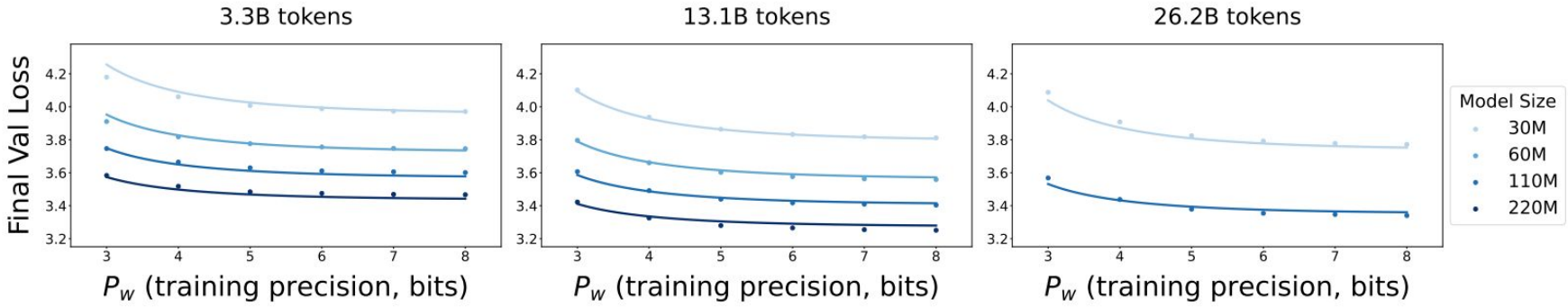
Insert into Chinchilla-style scaling:

$$L(N, D) = A[N(1 - e^{-P_{\text{w}}/\gamma_{\text{w}}})]^{-\alpha} + BD^{-\beta} + E$$

Interpretation

- Low bits help a lot at first
- Gains diminish at higher precision
- Around 6–7 bits, added bits give much less benefit

The simple N_eff fit predicts losses surprisingly well



Evidence

- Fit tested across multiple N, D, and Pw
- Predicted and empirical loss curves align well
- Same qualitative pattern across data budgets

Message

The weights-only case is not just intuition; it is quantitatively predictive.

Real low-precision training needs more than weight quantization

Why weights-only is incomplete

- Quantizing only weights does **not** give the main compute savings in pretraining
- For actual low-precision training, activations and attention/KV also matter

Next question

Can the same N_eff idea extend to:

- weights Pw
- activations Pa
- KV cache / attention P_{kv} ?

Finding 2: precision effects compose approximately independently

$$L(N, D, P_{\mathrm{w}}, P_{\mathrm{a}}, P_{\mathrm{kv}}) = A N_{\mathrm{eff}}^{-\alpha} + B D^{-\beta} + E$$

where

$$N_{\mathrm{eff}}(P_{\mathrm{w}}, P_{\mathrm{a}}, P_{\mathrm{kv}}) = N(1 - e^{-P_{\mathrm{w}}/\gamma_{\mathrm{w}}})(1 - e^{-P_{\mathrm{a}}/\gamma_{\mathrm{a}}})(1 - e^{-P_{\mathrm{kv}}/\gamma_{\mathrm{kv}}})$$

Empirical message

- Weights, activations, and KV precision each reduce effective capacity
- Their effects combine approximately **multiplicatively**
- Marginal fits and joint fits perform similarly

Why this matters

A very simple compositional law explains multi-part quantization behavior.

Same form, different sensitivity

Not all precisions are equally forgiving

- Weights are relatively tolerant
- Activations are more delicate
- Very low activation / KV precision can become unstable

Interpretation

Same functional form, but different fitted γ's

Practical reading

The bottleneck in pushing to ultra-low precision is often not the weights.

If training precision drops, increase parameters before data

At fixed compute

Lower precision reduces Neff

Compute-optimal response

- increase N
- decrease D relative to standard Chinchilla allocation

Intuition

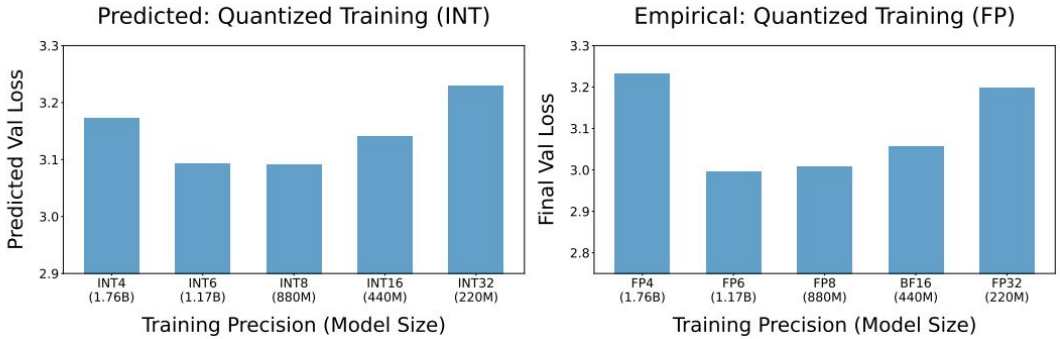
If precision shrinks effective capacity, adding more data alone becomes inefficient.

The paper's most provocative claim: mid precision may be optimal

Finding 3

When N,D,P are optimized jointly:

- compute-optimal training precision is approximately **independent of compute**
- fitted optimum is around **7–8 bits**



Interpretation

- 16-bit may be more than necessary
- below 4 bits may be too aggressive unless model size grows a lot

Fixed model size changes the answer

When N is fixed in advance

Optimal precision is no longer constant

$$P^*(C) \propto \log C.$$

Equivalent intuition:

for a fixed-size model trained on more and more data, higher precision can become more attractive.

Why

Precision becomes a lever for controlling overtraining relative to effective capacity.

A unified scaling law for training and inference precision

Two competing effects

- **Robustification:** low-precision training makes later PTQ less damaging
- **Overtraining effect:** lower training precision reduces N_{eff}}, which can increase PTQ sensitivity

Unified form

where

$$L(N, D, P_{\mathrm{w}}, P_{\mathrm{a}}, P_{\mathrm{kv}}, P_{\mathrm{post}}) = AN_{\mathrm{eff}}^{-\alpha} + BD^{-\beta} + E + \delta_{\mathrm{PTQ}}$$

- N_{eff}: training-time precision effect
- δ_{PTQ}: post-training quantization degradation

Main message

One framework captures both: low-precision pretraining effects & post-training inference-time degradation

Contributions, limitations, and final takeaways

What the paper contributes

- Treats **precision as a scaling variable**
- Models low-precision training via **effective parameter count**
- Unifies pretraining precision and PTQ in one law

Important caveats

- fixed architecture
- mainly studies **validation loss**, not downstream tasks
- most fits use **integer fake quantization**
- exact fitted constants likely depend on setup

Final takeaways

1. PTQ gets worse as models become more overtrained
2. Lower training precision reduces effective model capacity
3. Precision, parameters, and data should be chosen **jointly**
4. The trend is convincing; the exact numerical optimum should be treated cautiously

Backup slide 1 — PTQ overtraining trend is robust across quantizers

GPTQ (main text): degradation increases with token/parameter ratio

AWQ (Appendix F): same qualitative trend

RTN (Appendix F): same qualitative trend

Takeaway: “Overtrained models degrade more under PTQ” is not specific to one PTQ method

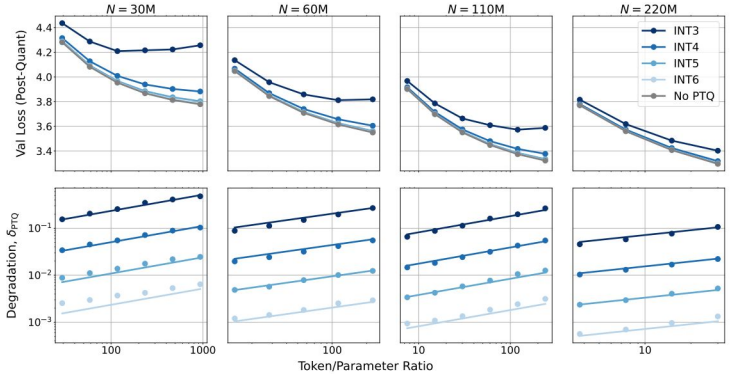


Figure 10: Replicating Section 3 results with AWQ.

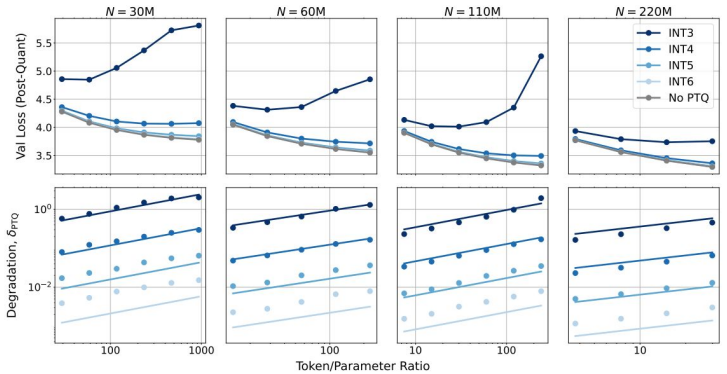


Figure 11: Replicating Section 3 results with RTN.

Main trends survive schedule and hyperparameter changes

Linear LR schedule ablation: PTQ sensitivity trend still appears

Hyperparameter ablations: early extra bits help a lot; gains then saturate

Dataset / training changes: qualitative precision trend remains

Takeaway: conclusions are not driven by one specific training recipe

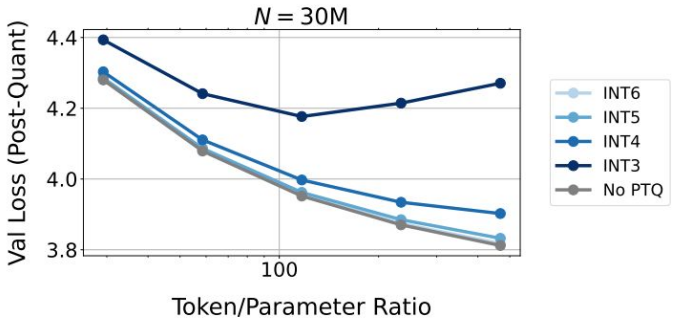


Figure 12: Linear LR Schedule Ablation

Two appendix nuances: component sensitivity and numeric format

A. Component sensitivity

- weights are more tolerant than activations / KV cache
- very low activation or KV precision can become unstable
- granularity ablation still suggests activations are harder

B. Floating-point vs integer

- main fits use integer quantization
- FP is harder because exponent and mantissa bits play different roles
- Appendix N shows similar qualitative trends in FP

Takeaway

The trend seems broader than one exact setup, but some components and formats are clearly harder than others.