# Scalable AI: Bridging Theory, Understanding, and Practice

Fall 2026

UC Berkeley

# Lecture 8: Introduction to Pre-Training

Objective, compute budgets, and training stability

Fall 2026

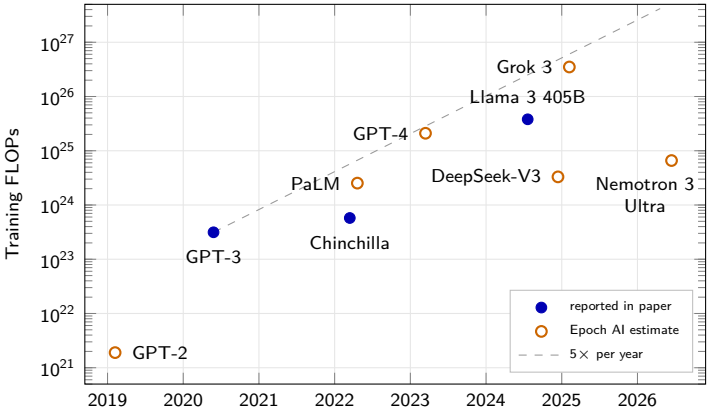
# Roadmap for today

| | |
|---|---|
| **1. Pre-training basics** | Causal language modeling objective, loss and perplexity, tokenization, loss curves |
| **2. The training step** | Batching, learning-rate schedules, optimizer setup, packing, mixed precision |
| **3. Compute and throughput** | FLOPs per token, time to train, the numbers to track daily |
| **4. Failures and stability** | Hardware failures and checkpointing; monitoring and evaluation; loss spikes and divergence |
| **5. Planning with small runs** | Hyperparameter tuning, model size, data mixture, stability checks; checklists |

# 1. Pre-Training Basics

Objective, tokenization, and what a loss curve looks like

# Pre-training compute over model generations




- About $10^5\times$ more compute from GPT-2 (2019) to Grok 3 (2025).
- Frontier models: about $5\times$ per year since 2020.
- Llama 3 405B used 30.8M H100-hours, roughly \$60–90M at \$2–3 per GPU-hour. Wasted compute is expensive.

Data: Epoch AI, "Data on AI Models" (accessed Sep 22, 2026). Estimates for closed models have wide error bars.

# Pre-training, mid-training, post-training

| Stage | What it is | **Olmo 3 32B Think** (share of GPU-hours) |
|---|---|---|
| **Pre-training** | Next-token prediction on a broad web-scale mixture | 5.5T tokens: 86% |
| **Mid-training** | Continued pre-training on higher-quality data while the learning rate decays | 200B tokens, incl. long context: 4% |
| **Post-training** | Instruction tuning (SFT), preference tuning, RL | SFT + DPO 1.6%, RL 7.9% |

- The split differs across models.
- Post-training is a growing share. Reasoning models are trained with RL to use more test-time compute. Olmo 3.1 ran RL three times longer than Olmo 3, to ≈16% of compute.

Olmo 3 (arXiv:2512.13961); Morrison et al., Olmo 3 compute accounting (arXiv:2605.01158), Table 4.

# Causal language modeling objective

## Autoregressive factorization

$$p_\theta(x_{1:L}) = \prod_{t=1}^{L} p_\theta(x_t \mid x_{<t})$$

## Maximum likelihood (cross-entropy) objective

$$\mathcal{L}(\theta) = -\frac{1}{N_{\text{tok}}} \sum_{x_{1:L} \sim \mathcal{D}} \sum_{t=1}^{L} \log p_\theta\left(x_t \mid x_{<t}\right)$$

Every token is the label for the prefix before it. No annotation is needed, so the training set is as large as the text you can collect and filter.

In practice, one forward pass with a causal attention mask gives all $L - 1$ predictions; the labels are the input shifted left by one.

# Cross-entropy and perplexity

$$\ell_t = -\log p_\theta(x_t \mid x_{<t}) \qquad \mathrm{PPL} = \exp\left(\frac{1}{N_{\mathrm{tok}}} \sum_t \ell_t\right)$$

Perplexity is the effective number of equally likely next tokens.

- Uniform guess over a 128K vocabulary: loss $\ln(128{,}000) \approx 11.8$, PPL $= 128$K.
- Loss 2.0: PPL $\approx 7.4$, as uncertain as a uniform choice among about 7 tokens.
- Loss values are only comparable with the same tokenizer and evaluation data.

# Tokenization

**The tokenizer determines**

- Tokens per document, so compute per document
- How many tokens each language or domain gets
- Handling of code whitespace, digits, rare scripts

Most models use byte-level BPE. Tokenize once, offline.

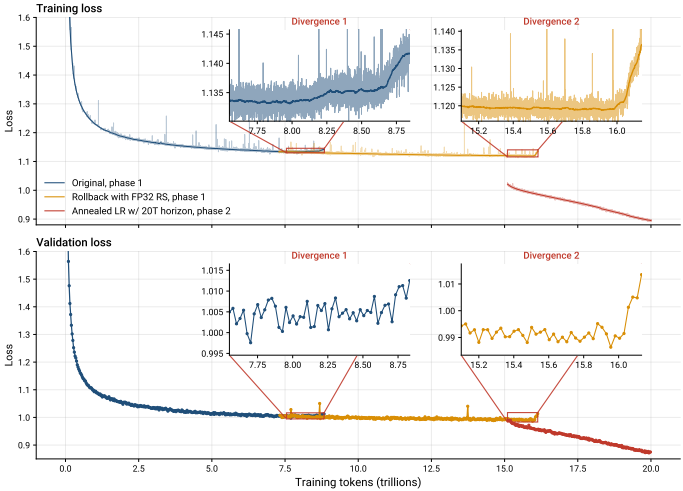
**Llama 2 → Llama 3**

| | Llama 2 | Llama 3 |
|---|---|---|
| Vocabulary | 32K | 128K |
| English characters per token | 3.17 | 3.94 |
| Tokens for the same text | 100% | ≈80% |

Cost: a larger embedding and output layer.

Llama 3 (arXiv:2407.21783) §3.2.

# What a pre-training loss curve looks like




Nemotron 3 Ultra, 20T tokens:

- Fast drop early, then a slow decline over trillions of tokens.
- Training loss is noisy; validation loss is smoother.
- Red: learning-rate decay on higher-quality data gives a further drop.
- Insets: two divergences (Section 4).

Figure: NVIDIA, Nemotron 3 Ultra technical report (arXiv:2606.15007), Fig. 5. CC BY 4.0.

# 2. The Training Step

Batching, learning rate and optimizer, packing, and mixed precision

# The training loop

```
for step, batch in enumerate(dataloader):
    tokens = batch["input_ids"] # [B, L]
    labels = batch["labels"] # [B, L], tokens shifted left by 1

    logits = model(tokens) # [B, L, |V|]
    loss = cross_entropy(logits, labels, ignore_index=PAD)

    loss.backward() # gradients
    clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step() # AdamW update
    scheduler.step() # learning rate
    optimizer.zero_grad(set_to_none=True)

    if step % ckpt_every == 0:
        save_checkpoint(model, optimizer, scheduler, dataloader, step)
```

At scale the same loop adds gradient accumulation, distributed collectives (Lecture 5), mixed precision, and activation checkpointing (recomputing activations during the backward pass to save memory).

# Choosing the batch size

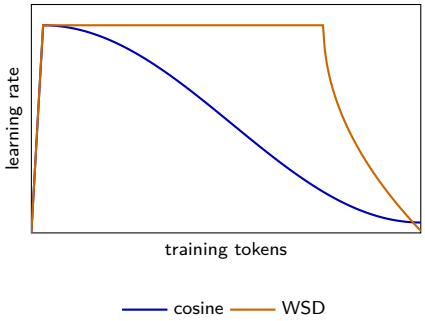
## Tokens per optimizer step

$$B_{\text{tok}} = \underbrace{R}_{\text{DP replicas}} \times \underbrace{B \times L}_{\text{microbatch}} \times \underbrace{A}_{\text{accumulation}}, \qquad \text{tokens/sec} = \frac{B_{\text{tok}}}{T_{\text{step}}}$$

- Gradient accumulation raises $B_{\text{tok}}$ without more activation memory.
- Larger batches use the hardware better: fewer optimizer steps and less communication per token. Past the **critical batch size**, more tokens per step give little extra progress, so tokens are wasted.
- The critical batch size grows as the loss falls, so runs start small and ramp up.
- Batch size and learning rate interact: change one and retune the other.
- Llama 3 405B: 4M $\rightarrow$ 8M $\rightarrow$ 16M tokens per step, with small batches early "to improve training stability."

Llama 3 §3.4.1; McCandlish et al., An Empirical Model of Large-Batch Training (arXiv:1812.06162).

# Learning-rate schedules




- **Warmup** avoids large updates while weights are poorly scaled. Llama 3 405B: 8,000 steps.
- **Cosine** decays over a fixed horizon (Llama 3).
- **WSD** (warmup, stable, decay) holds the rate, then decays at the end. The decay phase is often called the **cooldown**. Marin 8B, Nemotron 3 Ultra.
- With WSD you choose when to stop. Nemotron started its decay early after a divergence (Section 4).

Llama 3 §3.4.1; Marin 8B retrospective; Nemotron 3 Ultra §2.4.

# Setting up the optimizer

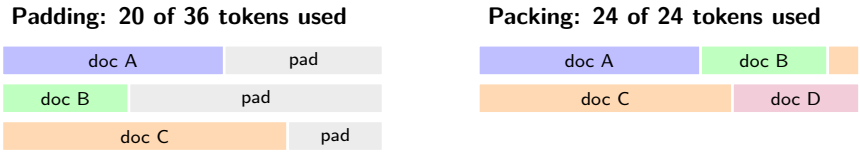
Many of the instabilities in Section 4 come from optimizer and parametrization settings: the learning rate, $\epsilon$, weight decay, initialization. These are much cheaper to get right before the run than to fix during it. Lecture 10 covers optimizers in detail.

**Common AdamW settings**

| | |
|---|---|
| $\beta_1, \beta_2$ | 0.9, 0.95 |
| $\epsilon$ | $10^{-8}$ |
| Weight decay | 0.1, not on embeddings |
| Gradient clipping | 1.0 |
| Peak learning rate | tuned; depends on width |

- The best learning rate shrinks as the model gets wider. With the standard parametrization, a rate tuned on a small model is too large for the full model.
- Use $\mu$P. It rescales per-layer learning rates and initialization so that settings tuned on a small model carry over to the full model (Section 5).
- OLMo 2 removed most of its spikes by changing initialization, $\epsilon$, and weight decay on embeddings (Section 4).

# Sequence packing




- Concatenate documents, separated by <EOS>, until each sequence holds $L$ tokens.
- No compute on padding, and every batch has the same shape.
- Optional document mask: attention stays within one document. Llama 3 found it mattered little in standard pre-training but helped for long-context training.

Llama 3 §3.2.

# Floating-point formats

Fewer bits per value means more tensor-core throughput and more rounding error. H100, dense: BF16 989 TFLOPs/s, FP8 1,979 TFLOPs/s.

| Format | Exponent bits | Mantissa bits | Largest | Step near 1 | 0.3952 is stored as |
|---|---|---|---|---|---|
| FP32 | 8 | 23 | $3.4 \times 10^{38}$ | $1.2 \times 10^{-7}$ | 0.39520 |
| BF16 | 8 | 7 | $3.4 \times 10^{38}$ | $7.8 \times 10^{-3}$ | 0.39453 |
| FP16 | 5 | 10 | 65,504 | $9.8 \times 10^{-4}$ | 0.39526 |
| FP8 E4M3 | 4 | 3 | 448 | 0.125 | 0.40625 |
| FP8 E5M2 | 5 | 2 | 57,344 | 0.25 | 0.375 |
| FP4 E2M1 | 2 | 1 | 6 | 0.5 | 0.5 |

- Exponent bits set the range; mantissa bits set the precision.
- FP16 needs loss scaling because small gradients underflow. BF16 has FP32's range and does not.
- FP8 and FP4 need an explicit scale per tensor or per block.

# Mixed precision

Each value gets the lowest precision it can tolerate.

## BF16: where speed comes from

- Inputs to matrix multiplies
- Activations saved for backward

## FP32: where errors accumulate

- Accumulation inside each GEMM
- Master weights, Adam moments, the update
- Softmax, normalization, the loss
- Gradient accumulation and reductions

**Why FP32 master weights.** Near 1.0, consecutive BF16 values are $2^{-7} \approx 0.008$ apart, so $1.0 + 0.001$ rounds to 1.0. Typical updates change a weight by $10^{-4}$–$10^{-3}$ of its value. Applied in BF16, most would round to zero.

The same applies to reductions. Llama 3's loss differed across parallelism layouts until gradient accumulation and reduce-scatter ran in FP32.

Llama 3 (arXiv:2407.21783) §3.3.2.

# Scaling low-precision values

$$x_{\text{low}} = \text{round}(x/s), \qquad \hat{x} = s \cdot x_{\text{low}}$$

The scale $s$ maps a group of values onto the format's small range.

| Format | Values | Values per scale | Scale format |
|---|---|---|---|
| FP8, per tensor (Hopper) | E4M3 / E5M2 | whole tensor | FP32, from recent max $\lvert x \rvert$ |
| MXFP8 (Blackwell) | E4M3 / E5M2 | 32 | power of two |
| NVFP4 (Blackwell) | E2M1 | 16 | E4M3, plus one FP32 scale per tensor |

- An outlier sets the scale for every value that shares it. Smaller blocks limit the damage, which is what makes 4-bit training possible.
- Nemotron 3 Ultra was pre-trained in NVFP4 on 20T tokens, with the last 15% of layers, attention projections, and embeddings kept in higher precision. Its loss stayed within 0.4% of BF16 runs branched from its checkpoints.

Transformer Engine FP8 primer; Nemotron 3 Ultra (arXiv:2606.15007) §2.2.

# 3. Compute and Throughput

FLOPs, time to train, and the numbers to track

# FLOPs per token

## Dense Transformer rule of thumb

$$\text{Training FLOPs} \approx 6\, N\, N_{\text{tok}}$$

Forward $\approx 2N$ per token, backward $\approx 4N$. $N$ = parameters; embeddings are about 1% of them at 405B, so the total is used below.

- Llama 3 405B: $6 \times 405\text{B} \times 15.6\text{T} = 3.79 \times 10^{25}$. Reported: $3.8 \times 10^{25}$.
- MoE: use active parameters. Nemotron 3 Ultra: $6 \times 55\text{B} \times 20\text{T} = 6.6 \times 10^{24}$.
- It undercounts long-context attention and recomputation from activation checkpointing.

Llama 3 §1; Nemotron 3 Ultra §1.

# From MFU to tokens per day

Time to train depends on how fast the cluster turns FLOPs into tokens.

$$\text{tokens/sec/GPU} \approx \frac{\text{MFU} \cdot C_{\text{peak}}}{6N}$$

MFU as in Lecture 3; $C_{\text{peak}}$ = peak FLOPs/s per GPU at the training precision.

$$\text{tokens/day} \approx G \cdot \text{goodput} \cdot \frac{\text{MFU}\, C_{\text{peak}}}{6N} \cdot 86{,}400$$

**Goodput** is the fraction of wall-clock time spent training. It is lost to data-loader stalls, checkpoint writes, evaluation, failures and restarts, and scheduling.

# Worked example: Llama 3 405B

**Reported**

| | |
|---|---|
| Parameters | **405B** (dense) |
| Tokens | **15.6T** |
| GPUs | **16,384** H100 |
| Throughput | **400 TFLOPs/s**/GPU |
| BF16 MFU | 38–43% |
| Compute used | **30.84M** H100-hours |

**Estimate**

$$\text{tokens/sec} \approx \frac{16{,}384 \times 400 \times 10^{12}}{6 \times 405 \times 10^{9}} \approx \mathbf{2.7 \times 10^{6}}$$

$$\text{time} \approx \frac{15.6 \times 10^{12}}{2.7 \times 10^{6}\ \text{s}^{-1}} \approx \mathbf{67\ days}$$

$$\text{GPU-hours} \approx 16{,}384 \times 67 \times 24 \approx \mathbf{26M}$$

26M of the reported 30.8M H100-hours (≈**85%**) went into model FLOPs at the stated rate. The rest is goodput loss and slower phases such as long-context training.

Llama 3 §3.3, Table 4; GPU-hours from the Llama 3.1 model card.

# Key performance indicators (KPIs)

KPIs are the handful of numbers a pre-training team reports every day.

| KPI | What it measures | Section |
|---|---|---|
| **Throughput**, MFU | Compute rate while the job runs | 3 |
| **Tokens/day** | Progress toward the token target | 3 |
| **Goodput** | Fraction of wall-clock time spent training | 3 |
| **Failures/day**, MTBF (mean time between failures) | How often the job stops | 4 |
| **Validation loss vs. prediction** | Validation loss against the prediction from small runs | 4, 5 |
| **Downstream evals** | Benchmarks and per-domain loss at each checkpoint | 4 |
| **Researcher interventions** | Manual actions needed to keep the run going | 4 |

Teams define these differently: write down the exact computation.

# Triage: which KPI moved?

$$\text{tokens/day} \approx \text{throughput while running} \times 86{,}400 \times \text{goodput}$$

**Throughput dropped**

- Kernel or shape change
- Lost communication overlap
- Stragglers
- Slow data loading

**Goodput dropped**

- More failures
- Slower restarts
- Slower checkpoints
- Preemption

**Validation loss off trend**

- Eval bug
- Data mixture change
- Schedule not restored after resume
- Precision change

When a KPI moves, first ask what changed: code, data, cluster, or schedule.

# 4. Failures and Stability

Hardware failures, monitoring, evaluation, and loss spikes

# Three ways a run fails

| | **The job stops** | **The loss diverges** | **Nothing looks wrong** |
|---|---|---|---|
| **Causes** | GPU, memory, network, and software faults | Learning rate, numerics, norm growth, bad data | Packing, masking, or label-shift bugs; wrong data mixture; data repeated after a resume; silent data corruption |
| **Noticed** | Immediately | From the loss, gradient norm, and auxiliary losses; can take hours | Only in evals and per-domain loss, sometimes after the run ends |
| **Recovery** | Restart from the last checkpoint, usually automatically | Roll back, change something, continue | Find the bug, then roll back to before it or restart |

Silent failures cost the most because they are found late. Tests before launch and evals during the run catch them.

# Hard failures at scale: Llama 3 405B

A 54-day window of Llama 3 405B pre-training. At this scale most lost goodput comes from failures.

| Cause (unexpected) | Count |
|---|---|
| Faulty GPU | 148 |
| GPU HBM3 memory | 72 |
| Software bug | 54 |
| Network switch or cable | 35 |
| Unplanned host maintenance | 32 |
| GPU SRAM | 19 |
| GPU system processor | 17 |
| NIC | 7 |
| NCCL watchdog timeout | 7 |
| Silent data corruption | 6 |
| Other (host, power, SSD, ...) | 22 |
| **Total** | **419** |

- 419 unexpected interruptions in 54 days, about 78% hardware.
- One every $\approx$3.1 hours.
- Per GPU: $\mathrm{MTBF} \approx 16{,}384 \times 3.1 \approx 50{,}000$ GPU-hours. At 100K GPUs: one every $\sim$30 minutes.
- Automation kept goodput above 90%. Only 3 interruptions needed significant manual intervention.

Llama 3 §3.3.4, Table 5 (counts; the table's printed percentages do not sum to 100).

# Checkpoint interval

Failures every $\tau$ hours, a checkpoint blocks training for $\delta$, checkpoint every $\Delta$:

$$\text{overhead} \approx \underbrace{\frac{\delta}{\Delta}}_{\text{writing}} + \underbrace{\frac{\Delta}{2\tau}}_{\text{lost work}} \quad \Rightarrow \quad \Delta^\star \approx \sqrt{2\,\delta\,\tau}$$

**Example:** $\tau = 3$ h, $\delta = 1$ min gives $\Delta^\star \approx 19$ min and $\approx$10% overhead. Asynchronous checkpointing shrinks $\delta$.

**A checkpoint contains**

- FP32 master weights
- Optimizer state
- Learning-rate scheduler state
- Data-loader position, RNG states

# Signals to log

## Optimization

- Training loss (raw and smoothed)
- Validation loss per domain; downstream evals
- Learning rate (actual, not configured)
- Gradient norm
- Update size $\|\Delta\theta\|$
- Auxiliary losses, e.g. MoE load balancing

## Numerics

- NaN/Inf checks on activations and gradients
- Activation and logit norms per layer

## Data

- Token statistics (length, language ID)
- Duplication and repetition rates
- Source mix of each batch

## Systems (on the same step axis)

- Step time, tokens/sec
- Data-loader wait
- Failure and restart events

# Evaluating a base model: validation loss

## Validation loss per domain

- Held-out sets for web, code, math, and each language, fixed for the whole run.
- The average hides regressions: code can get worse while the total improves.
- Training loss moves when the data mixture changes (OLMo 2's rose while the model improved). A fixed held-out set does not.

## Bits per byte

$$\mathrm{BPB} = \frac{N_{\mathrm{tok}}}{N_{\mathrm{bytes}}} \cdot \frac{\bar{\ell}}{\ln 2} \qquad \bar{\ell} = \text{mean loss per token, in nats}$$

Loss per token depends on the tokenizer; loss per byte does not. Example: loss 2.0 at 3.94 bytes per token (Llama 3, English) is $2.0/(3.94 \times 0.693) \approx 0.73$ bits per byte.

Paloma (arXiv:2312.10523); OLMo 2 (arXiv:2501.00656) §3.

# Evaluating a base model: benchmarks

## Scoring

- **Multiple choice:** compute the log-probability of each answer given the question, pick the highest. Normalize by answer length, since longer answers have lower totals. No generation needed.
- **Generative** (GSM8K, HumanEval): generate, then check the answer or run the tests.

## Format

- **Cloze:** score the answer text itself.
- **Letter:** list the options, score "A"–"D".
- Small models are at chance on the letter format. It starts to work only after enough training, so compare small models with the cloze format.

**Which benchmarks give signal early.** HellaSwag, ARC-Easy, PIQA, and similar cloze tasks rise above chance in small models. MMLU (letter format) and GSM8K stay near chance for much longer.

OLMES (arXiv:2406.08446); FineWeb (arXiv:2406.17557) §3.

# Evaluating a base model: reading the numbers

| Benchmark | Questions | Chance | SE at 50% |
|---|---|---|---|
| HellaSwag | 10,042 | 25% | 0.5 pts |
| MMLU | 14,042 | 25% | 0.4 pts |
| ARC-Easy | 2,376 | 25% | 1.0 pts |
| PIQA | 1,838 | 50% | 1.2 pts |
| ARC-Challenge | 1,172 | 25% | 1.5 pts |

Standard error $= \sqrt{p(1-p)/n}$.

- A 2-point gap on ARC-Challenge is within noise. Average over several tasks, or compare validation loss.
- Accuracy can sit at chance, then rise. The log-probability of the correct answer moves earlier and more smoothly.
- Number of shots, prompt format, and normalization move scores by several points. Fix them and record the exact configuration.
- Check for contamination when the data changes (Marin 32B: GSM8K in its cooldown data).
- Read samples: repetition, broken formatting.

OLMES (arXiv:2406.08446); Marin 32B retrospective.

# Gradient norm monitoring

$$\|g\|_2 = \|\nabla_\theta \mathcal{L}\|_2$$

For a fixed recipe, a stable run keeps its gradient norm in a characteristic range.

## What to watch

- Sudden spikes: often come before loss spikes
- Drops to 0: underflow or a communication bug
- Slow growth over the run: preceded divergence in OLMo-0424

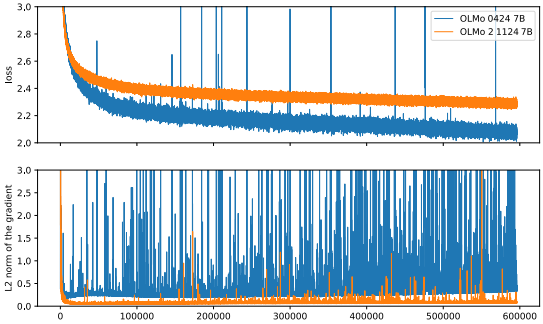
## How to log it

- Before clipping
- Globally and per layer
- Per parameter group (attention, MLP, embeddings, output)

**Spike score** (OLMo 2): percentage of values more than $7\sigma$ from the rolling mean of the last 1,000 steps.

OLMo 2 (arXiv:2501.00656) §3.

# Loss spikes and gradient-norm spikes




OLMo-0424 7B (blue) vs OLMo 2 7B (orange):

- Blue: gradient-norm spikes are more frequent than loss spikes and often come first.
- Blue: the gradient norm slowly grows, and spikes become more frequent.
- Orange: after changes to architecture, initialization, optimizer, and data, both curves are flat.

Figure: Ai2, OLMo 2 (arXiv:2501.00656), Fig. 2. CC BY 4.0. Orange loss is higher because the data changed.

# Divergence vs data spikes

## Divergence

- Loss rises and does not recover
- Gradient norm stays high or explodes
- NaNs/Infs may appear
- Often follows a learning-rate, precision, or code change

**Typical fixes:** roll back, lower the learning rate, undo the change, change the recipe.

## Data spike

- Isolated spike that returns to trend
- Gradient norm returns to its baseline
- Tied to particular batches or sources

**Typical fixes:** inspect the batch, filter the data, skip the batches if it recurs.

# Changes to the recipe: OLMo 2

| Component | Change | Gradient spike score |
|---|---|---|
| Initialization | Depth-scaled $\to \mathcal{N}(0, 0.02)$ | $0.40 \to 0.03$ |
| Normalization | Normalize each sublayer's output before the residual add; add QK-norm | $0.108 \to 0.069$ |
| Embeddings | No weight decay on embeddings | $0.16 \to 0.092$ |
| AdamW $\epsilon$ | $10^{-5} \to 10^{-8}$ | Lower, steadier gradient norm |
| Output logits | Add z-loss | Keeps softmax normalizers small |

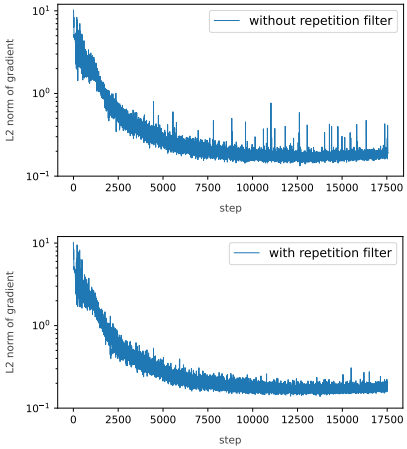
Each row is a separate ablation against its own baseline.

- **QK-norm**: normalize queries and keys before their dot product, so attention logits stay bounded.
- **z-loss**: a small penalty on $(\log Z)^2$, where $Z$ is the softmax normalizer of the output logits.

Most of these changes stop norms from growing without bound.

OLMo 2 (arXiv:2501.00656) §3, Table 1, Figs. 4–10.

# Spikes caused by data




- OLMo 2: batches at spikes held long repeated n-grams, e.g. `255, 255, 255, ...`
- Filter: drop documents with $\geq$32 consecutive repeats of an n-gram. Fewer gradient spikes (bottom).
- Not deterministic: the same sequence spiked a larger model but not a smaller one, and only under some data orders. Spikes depend on data *and* model state.
- Curated corpora such as Nemotron-CC remove most repetitive text. Marin 8B, trained mostly on DCLM and Nemotron-CC, saw very few spikes.

Figures: Ai2, OLMo 2 (arXiv:2501.00656), Fig. 3 and §3.1, CC BY 4.0; Nemotron-CC (arXiv:2412.02595); Marin 8B retrospective.

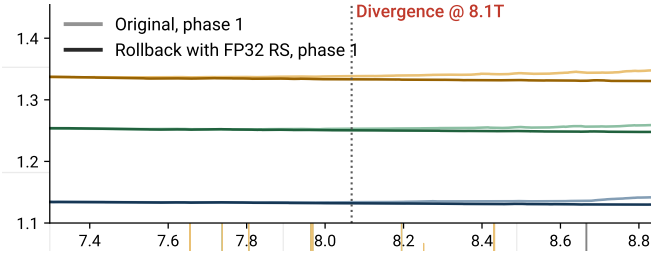
# Marin: spikes across three runs

| Run | Setup | Stability |
|---|---|---|
| Marin 8B | Llama architecture, 12.7T tokens, no QK-norm | Very few spikes |
| Marin 32B | Llama architecture, 6.4T tokens, no QK-norm | Frequent spikes. Clipping, skipping steps, and optimizer resets failed. Switching to QK-norm removed them |
| Marin 535B MoE (in progress) | QK-norm, z-loss | No spikes so far |

Their explanation: the 32B instability came from growing attention logits, not the optimizer. The stable 8B run made QK-norm look unnecessary.

Marin retrospectives: marin.readthedocs.io/en/latest/reports/ Code and W&B logs: github.com/marin-community/marin

# Nemotron 3 Ultra: two divergences




Loss (smoothed) vs. training tokens (T). Light: original run. Dark: rollback with FP32. Top to bottom: MTP-2, MTP-1, main loss.

**At 8T tokens** (figure)

- Output-layer gradient accumulation was moved from FP32 to BF16 to save communication.
- The multi-token prediction (MTP) heads, extra outputs that predict tokens further ahead, have small gradients. They were rounded away.
- The MTP-2 loss (top) rose before the main loss.
- Fix: roll back and restore FP32.

**At 16T tokens:** no cause found, and a BF16 rerun also diverged. Fix: roll back to 15T tokens and start the learning-rate decay. Pre-training ended at 20T.

Figure: NVIDIA, Nemotron 3 Ultra (arXiv:2606.15007), Fig. 6 and §2.7. CC BY 4.0.

# What to do when you see a spike

## Diagnose

1. Did the loss return to trend? Then record it and continue.
2. Check gradient norm, update norm, auxiliary losses.
3. What changed? Code, precision, data, cluster.
4. Inspect the batches around the spike.

## Recover

- Roll back and skip the batches
- Lower the learning rate
- Undo the recent change (Nemotron, 8T)
- If spikes keep coming, change the recipe (Marin 32B: QK-norm)
- Late in the run: start the decay (Nemotron, 16T)

For the next run, most of these fixes are chosen in advance and tested with small runs.

# 5. Planning with Small Runs

Choosing the recipe, size, and data before the full run

# Planning with small runs

Most decisions about a large run are made with small runs. In Olmo 3, 82% of GPU-hours went to development runs, not the final model.

A **ladder** is a set of models of increasing size trained with the same recipe. Llama 3 used 40M to 16B parameters to plan its 405B run.

| **Decision** | **How small runs decide it** |
|---|---|
| Learning rate, initialization | Tune on small models; $\mu$P makes them transfer |
| Batch size | Sweep on small runs; the best size grows with compute |
| Model size, token count | Fit a scaling law to the ladder |
| Data mixture | Ablations and anneals |
| Stability fixes | Run the ladder at the planned learning rate and batch size |
| Expected final loss | Extrapolate from the ladder; compare against it during the run |

Olmo 3 compute accounting: Morrison et al. (arXiv:2605.01158); Llama 3 §3.2.1; DeepSeek LLM (arXiv:2401.02954) §3.1.

# $\mu$P: hyperparameters that transfer across width

Recap from the deep learning course (prerequisite).

## Standard parametrization

- The best learning rate shrinks as width grows.
- A rate tuned on a small model is too large for a wide one: updates grow with width, logits grow, loss spikes.

## $\mu$P (Adam, width multiplier $m$)

- Hidden-weight learning rate $\div\, m$
- Output logits $\times\, 1/m$
- Attention scores scaled by $1/d_{\text{head}}$, not $1/\sqrt{d_{\text{head}}}$

Each layer's update then changes its output by the same amount at any width.

With $\mu$P, the learning rate and initialization tuned on a small model carry over to the full model. In the original paper, settings tuned on a 40M model beat the published 6.7B GPT-3, at 7% of its pre-training compute. $\mu$P covers width; depth needs extensions.

Yang et al., Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer (arXiv:2203.03466).

# Model size and token count

A compute budget $C \approx 6NN_{\text{tok}}$ fixes the product of parameters and tokens. How should it be split?

| Model | Params | Tokens | Tok./param |
|---|---|---|---|
| Gopher | 280B | 300B | 1 |
| Chinchilla | 70B | 1.4T | 20 |
| Llama 3 405B | 405B | 15.6T | 39 |
| OLMo 2 7B | 7B | 4T | 570 |
| Marin 8B | 8B | 12.7T | 1,600 |

- Gopher and Chinchilla used similar compute; the smaller model trained on more tokens was better. About 20 tokens per parameter is compute-optimal.
- Llama 3's ladder predicted 402B parameters and 16.55T tokens.
- Models meant for cheap inference are trained far past 20 tokens per parameter.

The budget has to cover the ladder, ablations, and anneals as well as the final run. Size the final run with what is left.

**Assignment 2** (released Sep 28): you will pre-train a model from scratch on one 8×H100 node for two weeks. Decide which ablations you want to run before the final run, and reserve compute for them.

Chinchilla (arXiv:2203.15556); Llama 3 §3.2.1; OLMo 2 §2.3; Marin 8B retrospective.

# Choosing the data mixture

| Run | Question | Small-run experiment and result |
|---|---|---|
| Marin | Pre-training mixture | Mixtures optimized on swarms of 300M models, then checked across scales |
| OLMo 2 | Mid-training data | 19 microanneals (130B tokens in total). Adding math data lifted a 7B model's GSM8K from 28.5 to 61 |
| Llama 3 | Pre-training mixture | Small models trained on candidate mixes, scaled with scaling laws. Final mix: 50% general, 25% math and reasoning, 17% code, 8% multilingual |

An **anneal** continues a partly trained model on candidate data while the learning rate decays. The effect shows up within tens of billions of tokens, far less than training from scratch.

Marin issue #6602; OLMo 2 §4.4; Llama 3 §3.1.2–3.1.3.

# Stability checks and a loss prediction

## Stability on the ladder

- Run the ladder at the planned learning rate and batch size
- Marin's ladder for its 535B run showed gradient-norm growth; adding z-loss fixed it before launch
- OLMo 2 reproduced its spikes in short small runs to compare initializations

## Loss prediction

- Marin pre-registers the full run's expected loss from the ladder
- During the run, loss above the prediction is a reason to investigate
- Llama 3 predicted ARC-Challenge accuracy across four orders of magnitude of compute; the final model scored slightly above the prediction

Small runs catch problems in the recipe. They do not catch every data spike: OLMo 2's repeated n-grams spiked a larger model but not a smaller one.

Marin issue #8435; OLMo 2 §3.1–3.2; Llama 3 §3.2.1.

# Checklists

## Essential

- Tests for tokenization, packing, and label shift
- Short runs to pick the learning rate (with $\mu$P so it transfers) and batch size
- Throughput measured and tuned before launch
- Checkpoint and resume test
- Training loss and gradient norm at every step
- Validation loss per domain on a fixed held-out set

## Beneficial

- A ladder for model size and a predicted final loss
- Benchmarks and samples at checkpoints
- Spike scores and per-layer norms
- Same loss across parallelism layouts
- Versioned data builds
- Goodput and failures tracked against the plan

# Summary

- Pre-training is causal language modeling on trillions of tokens. Frontier compute grows about $5\times$ per year.
- Mixed precision: multiply in low precision, sum in FP32. Low-precision sums caused real divergences.
- $6NN_{\text{tok}}$, MFU, and goodput give time-to-train: Llama 3 405B $\approx$ 67 days on 16K H100s.
- At 16K GPUs, a failure every few hours is normal. Checkpoint every $\approx \sqrt{2\delta\tau}$.
- Evaluate with per-domain validation loss in bits per byte, and cloze-format benchmarks read against their standard error.
- Silent bugs (packing, masking, data) cost the most; tests before launch and evals during the run catch them.
- Spikes come from the recipe (norm growth, numerics) and from data (repeated sequences). Gradient norms warn first; QK-norm, z-loss, and data filters address most of them.
- Plan with small runs: $\mu$P for hyperparameters, a ladder for size and a loss prediction, anneals for data. Budget compute for them before sizing the final run.