ChatGPT

Monitoring Metrics for LLM Pre-Training

Executive summary

LLM pre-training is the long-running process of optimizing a Transformer (dense or MoE) on next-token prediction over a very large token stream. The objective is simple; the *system* is not. A production pre-train run couples: optimizer dynamics and numerics (especially mixed precision), data ingestion/filtering/dedup, distributed efficiency (communication + stragglers), checkpointing/recovery, and a versioned evaluation harness. The highest-cost failures are often **silent**: training keeps consuming compute while learning stalls, stability degrades, throughput collapses due to stalls/stragglers, safety risk drifts upward, or benchmarks regress even though loss looks "normal." [1]

Primary sources highlight several metrics as unusually high-leverage:

- **Maximum attention logits** as a stability vital: Kimi K2 reports max attention logits can "quickly exceed 1000," typically producing loss spikes and occasional divergence; it caps attention logits with QK-Clip at $\tau = 100$. [2]
- **Attention entropy** as a sharpness/stability proxy: Apple's work tracks attention entropy per head, observes "entropy collapse" (near-zero entropy) correlates with oscillating loss or divergence, and notes that doubling learning rate can collapse entropy and cause divergence; it proposes σReparam to prevent collapse. [3]
- **Model FLOPs Utilization (MFU/HFU)** for normalized efficiency: PaLM reports 57.8% hardware FLOPs utilization and provides calculation details; MegaScale reports 55.2% MFU at 12,288 GPUs and argues deep observability is required because many stability issues only appear at scale. [4]
- **MoE routing health**: Switch Transformers formalizes expert capacity and a load-balancing loss and reports dropped tokens are typically <1% in their experiments. [5]
- **Data duplication and leakage**: the deduplication work finds >1% of unprompted LM output can be copied verbatim without dedup; dedup reduces memorization about 10× and train–test overlap affects >4% of validation sets in standard datasets. [6]
- **Toxicity regression probes**: RealToxicityPrompts (RTP) contains 100K prompts, ~22K with toxicity ≥ 0.5, and defines generation-based toxicity probability metrics. [7]
- **Log-first diagnosis**: L4 analyzes 428 production LLM training failures (May 2023–Apr 2024), finds hardware and user faults dominate, and reports current diagnosis relies heavily on training logs. [8]

For a **nano LLM training system**, the best return comes from a compact, robust "ICU panel" of per-step vitals plus periodic probes: (i) numeric health (NaN/Inf, loss scaling overflows/skips), (ii) optimization health (loss trend, grad/update scale), (iii) one or two architecture-specific internal vitals (max attention logit; optionally sampled attention entropy), (iv) throughput/latency decomposition, (v) basic data-mixture + dedup/leakage checks, and (vi) pinned evaluation artifacts. [9]

Pre-training monitoring playbook and formal model

What pre-training is

Let a model f_θ map context tokens x to a distribution over next tokens. Pre-training minimizes next-token loss over a time-varying data stream D_t (a mixture over sources/domains):

$$\min_\theta \ \mathbb{E}_{(x,y)\sim D_t}\big[\ell(f_\theta(x), y)\big].$$

An optimizer applies an update rule:

$$\theta_{t+1} = \theta_t - \eta_t \, \mathcal{A}\big(\nabla_\theta \ell_t\big),$$

where η_t is the learning rate schedule and $\mathcal{A}$ is an optimizer transform (e.g., AdamW moments). Mixed precision adds a numerical control loop (loss scaling) that may skip updates on overflow to avoid corrupting weights. [10]

Common challenges

- **Stability/numerics:** overflow/underflow, NaNs, loss spikes; attention sharpness pathologies such as attention-logit explosion and entropy collapse. [11]
- **Data quality/leakage:** duplicates, repeated substrings, train–val overlap, distribution drift. [6]
- **Efficiency/stragglers:** MFU regressions, dataloader stalls, communication bottlenecks, tail-latency stragglers. [12]
- **Safety/toxicity:** toxic degeneration and safety regressions due to data shifts; need checkpoint-level probes. [7]
- **Silent regressions:** benchmarks dip while "core training metrics" look fine; often caused by eval harness drift or data drift before architecture mismatch. [13]

Metric families and category-to-math mapping

This report groups metrics into seven categories. The categories align to components of the formal system:

- **Training/Optimization (architecture-agnostic):** probes the update dynamics $\theta_{t+1} = \theta_t - \eta_t \mathcal{A}(\nabla \ell_t)$ and the mixed-precision control loop.
- **Model Behavior (architecture-dependent):** probes internal computations of f_θ (attention logits/entropy; MoE routing), which are not defined for all architectures.
- **Data & Distribution:** probes the data stream D_t (mixture shares, drift statistics, duplicates, overlap).
- **Compute/Hardware:** probes time decomposition $T_{\text{step}} = T_{\text{input}} + T_{\text{compute}} + T_{\text{comm}} + T_{\text{other}}$ and MFU. [14]
- **Safety & Privacy:** probes risk-related functionals of data and sampled model outputs (toxicity probability, PII hits, memorization/regurgitation rates). [15]
- **Evaluation/Validation:** probes generalization under a fixed evaluation distribution and harness (val loss, slice metrics, microbench suites + artifact hashing).
- **Observability/Logging:** probes telemetry integrity and the diagnosis substrate (errors, restarts, checkpoint integrity, log-derived events). [8]

Why "Training/Optimization" is mostly architecture-agnostic while "Model Behavior" is architecture-dependent.
Training/Optimization metrics inspect the optimization boundary and numerics, which exist for nearly any gradient-trained model (loss, LR, gradients, optimizer state, AMP scaling). In contrast, Model Behavior metrics are defined by architectural mechanisms: attention logits and attention entropy exist because you have attention; MoE drop rate exists because you route tokens to experts. Kimi K2's max-attention-logit monitoring and Apple's attention-entropy monitoring are explicit examples of architecture-dependent "internal vitals." [16]

Short diagnostic workflow

When an alert triggers:

1. **Contain corruption:** NaN/Inf, checkpoint failures, sustained overflow/skip storms.
2. **Localize the stack:** optimization/numerics ↔ model behavior ↔ data drift ↔ hardware/stragglers.
3. **Replay invariants:** fixed probe batch + pinned evaluation artifacts (hashes).
4. **If benchmarks regress:** eval harness → data drift → internal behavior → architecture. [17]

Monitoring pipeline and incident response:

```
flowchart LR
  subgraph Train["Training job"]
    A["Trainer vitals\n(loss/LR/grad/update)\nprogress counters"]
    B["Internal probes\n(attn logits/entropy,\nMoE routing)"]
    C["Data probes\n(mix/lang/filter,\ndedup/leak)"]
    D["System telemetry\n(step time/tokens/s,\nMFU, comm/input)"]
    E["Events/logs\n(OOM/NCCL/IO)\nrestarts/ckpt errors"]
  end

  A --> M["Typed metrics stream"]
  B --> M
  C --> M
  D --> M
  E --> L["Structured logs"]

  M --> TS["Time-series DB"]
  L --> LS["Log store"]
  TS --> Dash["Dashboards"]
  LS --> Dash
  TS --> Alert["Alerts + runbooks"]
  LS --> Alert
  Alert --> Train
```

```
flowchart TD
  S["Alert fires"] --> Sev{"SEV0 / SEV1 / SEV2"}
```

```
  Sev -->|SEV0\nNaN/Inf, ckpt fail,\noverflow storm| C0["Pause/stop\nprotect
last-good ckpt\nsnapshot evidence"]
  Sev -->|SEV1\nloss spikes,\nattn eruption,\nMFU collapse| C1["Contain\nreduce
LR / enable guards\nincrease sampling\nprepare rollback"]
  Sev -->|SEV2\ndrift, stalls,\nslow regressions| C2["Investigate\nlocalize
shard/node\nrun canary"]

  C0 --> Dx["Diagnose:\ncorrelate metrics with\nshard + rank + logs"]
  C1 --> Dx
  C2 --> Dx
  Dx --> Fix["Fix + verify:\nreplay probe batch\nrerun micro-eval\nresume or
rollback"]
```

Metric catalog by category

Each category starts with a short definition. Each table includes: name, category, definition, units, how to compute, typical ranges (small/medium/large; if unknown: “unspecified”), sampling frequency, suggested alert thresholds, and primary data sources.

Training and optimization metrics

Category definition. Metrics that measure whether gradient-based learning is proceeding correctly and whether mixed precision is applying updates safely. These metrics mostly depend on $\theta_{t+1} = \theta_t - \eta_t \mathcal{A}(\nabla \ell_t)$ and the AMP loss-scaling controller, not on attention vs MoE details. [18]

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	s a t
tokens_seen	Training/ Optimization	cumulative effective (non-pad) target tokens processed	tokens	sum of non-pad target tokens across microbatches	unspecified	step	A s
steps_seen	Training/ Optimization	successful optimizer updates applied	steps	increment after optimizer step that updates weights	unspecified	step	s w c
train_loss_raw	Training/ Optimization	batch cross-entropy	nats/ token	mean NLL over non-pad targets	unspecified	step	l s c

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling
train_loss_ema	Training/ Optimization	smoothed training loss	nats/ token	EMA(β≈0.95–0.999)	unspecified	step
train_ppl	Training/ Optimization	perplexity proxy (exp loss)	unitless	`exp(train_loss_ema)`	unspecified	step/ window
learning_rate	Training/ Optimization	effective LR (per param group)	unitless	scheduler state	unspecified	step
lr_phase	Training/ Optimization	warmup/ decay progress	%	step / warmup_steps etc.	unspecified	10–100 steps
tokens_per_update	Training/ Optimization	effective global batch size (tokens per update)	tokens/ update	sum effective tokens across grad-accum window	unspecified	step
grad_global_norm	Training/ Optimization	global L2 norm of gradients after AMP unscale	unitless	sqrt(sum(g^2))	unspecified	step
grad_clip_rate	Training/ Optimization	fraction of updates with clipping active	%	clipped_steps / steps (window)	unspecified	window
update_to_weight	Training/ Optimization	relative update magnitude ‖Δθ‖/‖θ‖	unitless	compute update norm vs param norm (sample layers ok)	unspecified	10–100 steps

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling
opt_state_finite	Training/ Optimization	optimizer moments/ states are finite	bool/%	isfinite checks on sampled state tensors	should be finite	1k–10k steps
loss_scale	Training/ Optimization	AMP loss scale	unitless	scaler state	unspecified; NVIDIA reports constant scales 8–32K in examples	step
overflow_rate	Training/ Optimization	overflow frequency in window	%	overflow_events / steps in window	~0 after warmup	window
skipped_updates	Training/ Optimization	skipped optimizer updates due to overflow	count/ %	“found_inf” → skip update	~0 after warmup	step/ window
grad_noise_scale	Training/ Optimization	gradient noise scale estimate	unitless	McCandlish estimator w/ gradient probes	unspecified	hourly/ daily
scaling_law_residual	Training/ Optimization	deviation from expected loss-vs-tokens curve	nats/ token	fit power-law curve; track residuals	unspecified	daily
curvature_trace_est	Training/ Optimization	curvature proxy via Hessian trace estimator	unitless	Hutchinson: avg($v^T Hv$) via HVP	unspecified	daily/ weekly

Model behavior metrics

Category definition. Metrics that measure internal computations of f_θ, especially attention and MoE routing. These diagnose failures where loss looks stable but internals drift into known-bad regimes (sharpness, routing collapse). [23]

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	s a t
max_attn_logit	Model Behavior	max pre-softmax attention score (per head or global)	logit	max(QKᵀ/√d_head)	Kimi reports >1000 in unstable regime; τ=100 used as cap	step/ sampled	v c
attn_entropy	Model Behavior	entropy of attention probabilities per head	nats	H=-Σ p log p	unspecified	10–100 steps	b c t
residual_rms	Model Behavior	residual stream RMS per layer	unitless	forward hooks; per-layer p50/p99	unspecified	10–100 steps	l r
activation_outlier_rate	Model Behavior	fraction of activations beyond threshold	%	count(	a	>k·σ) per layer	u
logit_entropy_probe	Model Behavior	entropy of output distribution on fixed probe	nats	entropy(softmax(logits))	unspecified	100–1k steps	s c
topk_mass_probe	Model Behavior	top-k probability mass on probe	%	Σ_topk p	unspecified	100–1k steps	j c
output_kl_to_prev	Model Behavior	KL drift vs previous checkpoint on probe	nats	KL(p_t‖p_{t-1}) on fixed probe	unspecified	per checkpoint	s

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	s a t
moe_drop_rate	Model Behavior	tokens dropped due to expert overflow	%	dropped/routed	Switch: typically <1%	step/ window	s > >
moe_util_entropy	Model Behavior	entropy (or CV) of expert token counts	unitless	entropy(counts) or CV(counts)	unspecified	window	e c
moe_load_balance_loss	Model Behavior	aux loss encouraging balanced routing	unitless	Switch aux loss terms	unspecified	step	r i

Data and distribution metrics

Category definition. Metrics that characterize the evolving training distribution D_t and its quality: mixture shares, filtering yield, length/encoding anomalies, duplicates, leakage, and drift. [27]

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	s a t
source_mix_share	Data & Distribution	token share per data source/ domain	%	count non-pad tokens by source tag	unspecified	hourly/daily	c
language_share	Data & Distribution	token share per language	%	language ID per doc/ chunk	unspecified	daily	u s
filter_reject_rate	Data & Distribution	reject fraction per filter stage	%	rejected/total per rule	unspecified	daily	s c

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	s a t
bytes_per_token	Data & Distribution	bytes/ token (junk/ encoding proxy)	bytes/ token	len_utf8(text) / tokens	unspecified	daily	l s
seq_len_p50_p95	Data & Distribution	length distribution of sequences	tokens	percentiles over batches/docs	unspecified	daily	a
pad_fraction	Data & Distribution	padding tokens fraction	%	pad / total tokens	unspecified	step/window	s >
token_kl_drift	Data & Distribution	token histogram KL drift vs baseline	nats	KL(hist_now||hist_ref)	unspecified	hourly/daily	s
near_dup_rate	Data & Distribution	within-train near-duplicate fraction	%	MinHash/LSH dedup	dedup paper: >1% unprompted verbatim emission w/ o dedup	build+periodic	r d
train_val_overlap	Data & Distribution	near-dup overlap train↔val	%	near-dup join	>4% overlap in standard datasets	build-time	a t d
doc_repeat_hotspots	Data & Distribution	large repeated substrings/ clusters	counts	cluster size distribution	unspecified	build-time	g d

Compute and hardware metrics

Category definition. Metrics that characterize time and resource efficiency: throughput, step latency, decomposition (input/compute/comm), tail-latency stragglers, and MFU/HFU. [4]

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	sugg alert thres
tokens_per_sec	Compute/ Hardware	effective throughput	tokens/ s	effective_tokens / wall_time	unspecified	step + 1m avg	drop 20%
step_time	Compute/ Hardware	wall time per optimizer update	s	timer around update	unspecified	step	heav
mfu	Compute/ Hardware	model FLOPs utilization	%	tokens/s × FLOPs/ token / peak_FLOPs	PaLM HFU 57.8%; MegaScale MFU 55.2%	10–60s avg	susta drop
gpu_util	Compute/ Hardware	GPU utilization	%	NVML/DCGM	unspecified	10–60s	low u high step_
gpu_mem_used	Compute/ Hardware	GPU memory usage	GB/%	NVML + framework	unspecified	10–60s	creep near-
dataloader_stall_frac	Compute/ Hardware	fraction of step waiting on input	%	time_in_input / step_time	unspecified	10–60s avg	susta >10–1
comm_time_frac	Compute/ Hardware	fraction in collectives	%	time_in_comm / step_time	unspecified	10–60s avg	rising trend
compute_time_frac	Compute/ Hardware	fraction in compute kernels	%	time_in_compute / step_time	unspecified	10–60s avg	sudd drop
straggler_ratio	Compute/ Hardware	tail rank slowdown (max/ median)	ratio	max(step_time_rank)/ median(step_time_rank)	unspecified	10–60s avg	susta high

Safety and privacy metrics

Category definition. Metrics that detect safety regressions and privacy risks: toxicity in data and outputs, PII presence, and memorization/regurgitation. These are usually checkpoint-level probes with stable artifact versioning. [15]

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	suggested alert thresholds	prima data sourc
data_toxicity_rate	Safety & Privacy	toxicity score stats on training-data sample	score/ %	run toxicity classifier on sampled docs	unspecified	daily	spike vs baseline	data samp [30]
rtp_toxicity_prob	Safety & Privacy	probability of toxic generation on RTP	%	P(tox≥0.5 at least once in k gens)	unspecified	per checkpoint	regression beyond band	mode gen + RTP
rtp_prompt_stats	Safety & Privacy	RTP dataset stats (100K, ~22K toxic)	counts/ %	dataset property	fixed	N/A	N/A	RTP
pii_hit_rate	Safety & Privacy	PII detector hit rate	%	detectors over data and sampled outputs	unspecified	build + checkpoint	any high-confidence spike	pipel + probe
memorization_rate	Safety & Privacy	verbatim match rate to training data	%	hashing / suffix match vs train index	>1% unprompted verbatim w/ o dedup; ~10× less with dedup	per checkpoint	rising trend	gen probe [28]

Evaluation and validation metrics

Category definition. Metrics that measure generalization on held-out distributions and task probes, with strict harness and artifact control. [31]

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	suggested alert thresholds
val_loss	Evaluation/ Validation	held-out cross-entropy	nats/ token	eval forward pass	unspecified	per checkpoint	regression beyond band
val_ppl	Evaluation/ Validation	exp(val_loss)	unitless	exp(val_loss)	unspecified	with val_loss	jump
train_val_gap	Evaluation/ Validation	generalization gap	nats/ token	val_loss − train_loss_ema	unspecified	with val_loss	widening trend
slice_val_loss	Evaluation/ Validation	loss by slice (domain/lang/ length)	nats/ token	stratified eval	unspecified	per checkpoint	slice regression
fixed_probe_loss	Evaluation/ Validation	loss on frozen probe batch	nats/ token	eval on immutable tokens	unspecified	100–1k steps	discontinui
microbench_score	Evaluation/ Validation	small benchmark suite scores	mixed	pinned harness	unspecified	per checkpoint	regression
eval_artifact_hash	Evaluation/ Validation	hash of tokenizer/ prompts/ decoding	id/hash	hash evaluation artifacts	must match	every eval	mismatch invalidates diffs

Observability and logging metrics

Category definition. Metrics that ensure you can detect, localize, and recover: NaN/Inf events, failures/ restarts, checkpoint integrity, telemetry lag, and structured errors. L4's findings motivate treating logs and diagnosis data sources as first-class. [8]

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	suggested alert thresholds
nan_inf_events	Observability/ Logging	NaN/Inf in loss/ grads/ weights	count	isfinite checks	expected 0	step	any >0 → SEV0

name	category	definition	units	how to compute	typical ranges (small / medium / large)	sampling	suggested alert thresholds
checkpoint_ok	Observability/ Logging	checkpoint save + verification success	bool	atomic write + checksum + restore test	should be true	checkpoint	any failure → SEV0
checkpoint_time	Observability/ Logging	time to save checkpoint	s	timer	unspecified	checkpoint	spike/ growth
checkpoint_size	Observability/ Logging	bytes written per checkpoint	GB	file size	unspecified	checkpoint	sudden jump
restart_rate	Observability/ Logging	job restarts per hour/ day	count/ time	orchestrator events	unspecified	continuous	storm
error_event_rate	Observability/ Logging	errors by class (OOM/ NCCL/IO)	count/ time	structured log parsing	unspecified	continuous	spike/ repeat
metrics_ingest_lag	Observability/ Logging	delay from emission to visibility	s	now − metric timestamp	unspecified	continuous	>minutes
heartbeat_missing	Observability/ Logging	missing rank/node heartbeats	count	heartbeat timeouts	should be 0	continuous	any burst

Per-metric playbooks

Each playbook provides: (1) intuition, (2) why it matters, (3) good/bad patterns, (4) examples from primary sources (when available), (5) common failure modes, (6) concrete mitigations.

To keep this report usable, “good/bad patterns” emphasize **trend shape** and **couplings** between metrics, not just absolute thresholds (which are often workload-specific).

Training and optimization playbooks

tokens_seen

Intuition. A monotone “true progress” counter: how much training signal you have consumed.
Why it matters. GPUs can be busy while training is not progressing (stalls, retries, repeated skips). tokens_seen is the simplest ground-truth progress witness in long jobs. [29]
Good/bad patterns. Good: steady increase with stable slope. Bad: flatline; sawtooth resets after restarts; stepwise slope drops.
Example. MegaScale frames long-job stability as maintaining efficiency and progress; progress counters are foundational to any observability stack. [29]
Failure modes. Dataloader deadlock; input pipeline stall; distributed hang; repeated overflow skips preventing updates. [33]
Mitigations. Alert if Δtokens=0 for K steps; snapshot last N batch metadata (source, lengths); check dataloader_stall_frac and error_event_rate; if overflow-related, inspect loss_scale and overflow_rate and reduce LR. [33]

steps_seen

Intuition. Count of applied optimizer updates (not microsteps).
Why it matters. AMP can skip updates when overflows occur; you may “run” but not learn. NVIDIA describes skipping weight updates on overflow and adjusting scaling factors dynamically. [10]
Good/bad patterns. Good: steps increase proportionally to tokens_seen and accumulation settings. Bad: steps stall while tokens grow (skips); steps reset on resume.
Example. NVIDIA notes dynamic loss scaling skips weight updates when overflow occurs and adjusts scale; skips should be infrequent in steady training. [19]
Failure modes. Persistent overflows; unscale/clip ordering bugs; optimizer exceptions; resume mismatch.
Mitigations. Treat sustained skips after warmup as SEV1/SEV0; reduce LR; verify unscale-before-clip; lower initial loss_scale or tune dynamic scaling parameters. [19]

train_loss_raw and train_loss_ema

Intuition. Instantaneous vs smoothed objective value.
Why it matters. Loss spikes often precede divergence; drift upward can indicate data drift or optimizer corruption. Kimi K2 explicitly ties attention-logit explosion to loss spikes/instability and introduces QK-Clip to stabilize training. [24]
Good/bad patterns. Good: smooth decrease then plateau; EMA smooth. Bad: spike clusters; oscillation; monotone drift up.
Example. Kimi K2 reports that max attention logits >1000 “usually result in instability … including significant loss spikes and occasional divergence.” [34]
Failure modes. LR too high; outlier shard; attention instability; overflow events; data distribution shift. [35]
Mitigations. On spike: snapshot max_attn_logit, grad_global_norm, overflow_rate, and batch source tags; quarantine shard if spikes are source-local; stabilize attention (QK-Clip-like) and reduce LR if coupled to attention logits. [36]

```
xychart-beta
  title "Train loss patterns (illustrative)"
```

```
x-axis "progress" [0, 1, 2, 3, 4, 5]
y-axis "loss (nats/token)" 0 --> 6
line "healthy" [5.1, 3.9, 3.1, 2.7, 2.5, 2.4]
line "spiky"   [5.1, 3.9, 3.2, 3.8, 2.9, 3.6]
```

train_ppl

Intuition. A monotone transform of loss: $\exp(\text{loss})$.
Why it matters. Makes multiplicative regressions easy to see ("perplexity doubled").
Good/bad patterns. Good: decreases or plateaus. Bad: jumps ×2+ often indicate pipeline or numerical issues.
Example. Use alongside val_ppl to separate optimization noise from generalization drift; scaling-law papers interpret loss curves quantitatively. [21]
Failure modes. Tokenization mismatch; masking bug; overflow causing corrupted updates.
Mitigations. Replay fixed_probe_loss; validate eval_artifact_hash and tokenizer hash; check overflow_rate and nan_inf_events. [33]

learning_rate and lr_phase

Intuition. The step-size schedule and where you are in it.
Why it matters. Sharpness/stability depend strongly on LR. Apple reports attention entropy collapse and divergence when hyperparameters such as LR are not tuned; doubling LR collapses entropy to near zero and diverges in experiments. [37]
Good/bad patterns. Good: continuous schedule; correct resume. Bad: discontinuities at restart; wrong warmup phase.
Failure modes. Scheduler state not checkpointed; global step mismatch; config drift.
Mitigations. Checkpoint scheduler state; on resume assert LR continuity; if instability starts at schedule transitions, rerun canary with longer warmup or lower peak LR and monitor attention entropy/logits. [38]

tokens_per_update

Intuition. Effective global batch size measured in tokens, not sequences.
Why it matters. Changes optimization regime and invalidates comparisons; interacts with gradient noise scale and scaling laws. [39]
Good/bad patterns. Good: stable. Bad: drift after packing/length changes.
Failure modes. Packing/bucketing bug; accumulation miscount; adaptive microbatching on OOM.
Mitigations. Treat as an invariant; alert on drift; gate batcher changes with canaries; if changed intentionally, re-tune LR. [40]

grad_global_norm

Intuition. Gradient magnitude after correct AMP unscale.
Why it matters. Exploding gradients precede divergence; AMP's overflow detection hinges on NaN/Inf gradients. [19]
Good/bad patterns. Good: stable band. Bad: spikes or monotone growth; spikes correlated with a shard or max_attn_logit.
Failure modes. LR too high; attention instability; outlier batches; wrong unscale/clip ordering.

Mitigations. Verify unscale-before-clip; reduce LR; if spikes correlate with attention internals, stabilize attention (QK-Clip or σReparam-like control). [41]

grad_clip_rate

Intuition. Fraction of steps where clipping is activated.
Why it matters. Persistent clipping is a symptom: you are constantly hitting a safety rail, often with learning cost.
Good/bad patterns. Good: rare after warmup (unless intended). Bad: rising trend; sustained high fraction.
Failure modes. Outlier data; LR too high; internal sharpness issues.
Mitigations. Use it as a symptom, then localize: shard id, attention logits, overflow; reduce LR; improve filtering; stabilize attention. [36]

update_to_weight

Intuition. Relative update size $||\Delta\theta||/||\theta||$, a direct "effective step size" proxy.
Why it matters. Catches optimizer-state corruption and resume mismatch even when configured LR looks fine.
Good/bad patterns. Good: stable with planned schedule transitions. Bad: sudden jump at resume; layer-local spike.
Failure modes. Optimizer state mismatch; weight decay misapplied; AMP skip/step bugs.
Mitigations. Validate opt_state_finite; compare with prior runs; rollback if jump coincides with val regression. [33]

opt_state_finite

Intuition. "Are optimizer states numerically valid?"
Why it matters. Corrupted moments can cause slow drift and late regressions; L4 reports diagnosis relies heavily on logs because hidden failures and corrupt states are otherwise hard to localize. [8]
Good/bad patterns. Good: always finite. Bad: any NaN/Inf.
Failure modes. Partial checkpoint; dtype conversion bug; offload error.
Mitigations. SEV0 rollback; add checksums and schema versioning for optimizer state. [42]

loss_scale, overflow_rate, skipped_updates

Intuition. Mixed precision control-loop state. NVIDIA explains constant scaling factors (8 to 32K in tested networks) and describes dynamic scaling: if overflow occurs, skip weight update and reduce scale; after N non-overflow iterations, increase scale. [10]
Why it matters. You can waste compute without learning if updates are repeatedly skipped; persistent overflows often correlate with instability.
Good/bad patterns. Good: early occasional overflow; later overflow_rate ~0 and loss_scale stabilizes. Bad: sustained non-zero overflow_rate; loss_scale collapses; steps_seen lags tokens_seen.
Failure modes. LR too high; unsafe ops in FP16; attention instability causing extreme gradients.
Mitigations. Reduce LR; keep numerically unsafe ops in FP32; stabilize attention; if overflow persists, pause and rollback to avoid corrupting weights. [35]

```
xychart-beta
  title "Overflow rate (illustrative)"
  x-axis "progress" [0, 1, 2, 3, 4, 5]
  y-axis "overflow rate (%)" 0 --> 20
  line "healthy" [12, 3, 1, 0, 0, 0]
  line "bad"     [12, 10, 12, 15, 14, 13]
```

grad_noise_scale

Intuition. A statistic predicting the largest useful batch size; McCandlish et al. report it increases as loss decreases and describes compute-efficiency vs time-efficiency tradeoffs and adaptive batch sizing. [20]
Why it matters. Guides batch scaling decisions, especially when you want to trade tokens_per_update for wall-clock speed without harming token efficiency.
Good/bad patterns. Good: smooth evolution. Bad: discontinuity after data/hyperparameter changes (often drift or estimator noise).
Failure modes. Noisy estimator; changing data mixture; optimizer differences.
Mitigations. Compute periodically on controlled probes; treat as guidance not an automatic controller in nano systems. [20]

scaling_law_residual

Intuition. Compare observed loss-vs-tokens to a fitted scaling-law curve; track residuals.
Why it matters. Kaplan et al. report loss scales as a power law with model size, dataset size, and compute; Hoffmann et al. report compute-optimal training requires scaling tokens with model size (Chinchilla). Deviations can indicate data quality drift or under/over-training relative to compute. [21]
Good/bad patterns. Good: residuals fluctuate around 0 within a band. Bad: step-change residual drift after data/pipeline changes.
Failure modes. Data contamination; dedup changes; optimizer bugs; batch size drift.
Mitigations. If residual jumps, check data drift metrics and eval invariants; run a canary to confirm; consider retracing LR/batch scaling. [43]

curvature_trace_est

Intuition. A curvature proxy: approximate $\mathrm{Tr}(H)$ with Hutchinson's estimator $\mathbb{E}[v^\top H v]$ using Hessian-vector products. [22]
Why it matters. Sharpness increases often precede instability and can explain "loss looks fine but generalization regresses."
Good/bad patterns. Good: stable or slowly varying. Bad: sharp rise aligned with instability or validation regression.
Failure modes. Too-large LR; entering sharp minima; numeric issues amplified by scale.
Mitigations. Schedule LR down; increase regularization; validate on probes; keep this metric sparse (daily) for nano systems due to cost. [44]

Model behavior playbooks

max_attn_logit

Intuition. The maximum pre-softmax attention score, typically $S = QK^\top/\sqrt{d}$.
Why it matters. Kimi K2 reports max attention logits can exceed 1000 and this "usually result[s] in instability ... including significant loss spikes and occasional divergence"; it uses QK-Clip with τ=100 to cap logits. [24]
Good/bad patterns. Good: bounded max; shrinking tail; stable across restarts. Bad: monotone growth; large fraction of heads exceeding thresholds.
Failure modes. Attention-logit explosion; LR too high; unstable optimizer regime.
Mitigations. Add per-head monitoring; cap logits (QK-Clip-like); reduce LR; correlate with entropy collapse; rollback if coupled with NaNs. [45]

```
xychart-beta
  title "Max attention logit (illustrative)"
  x-axis "progress" [0, 1, 2, 3, 4, 5]
  y-axis "max attn logit" 0 --> 1200
  line "bounded (tau-like)" [100, 100, 95, 90, 85, 80]
  line "runaway"            [120, 250, 500, 850, 1100, 1150]
```

attn_entropy

Intuition. Entropy of attention probability vectors; low entropy means extremely concentrated attention ("sharpness").
Why it matters. Zhai et al. track attention entropy per head, identify "entropy collapse" as a common pattern where low entropy accompanies instability, and show doubling LR collapses entropy near zero and diverges. They also prove a lower bound on attention entropy that decays exponentially with the spectral norm of the attention logits, motivating spectral control. [25]
Good/bad patterns. Good: stable entropy distributions (some heads may be sharp, but not broad collapse). Bad: widespread drift toward 0; collapse coupled to LR or max_attn_logit increase.
Failure modes. Logit explosion; hyperparameter mis-tuning; unchecked growth of Q/K spectral norms.
Mitigations. Reduce LR; consider spectral control (σReparam concept); monitor max_attn_logit and residual_rms; increase head/layer sampling during incidents. [46]

```
xychart-beta
  title "Attention entropy collapse (illustrative)"
  x-axis "progress" [0, 1, 2, 3, 4, 5]
  y-axis "entropy (nats)" 0 --> 3
  line "healthy"  [2.2, 2.0, 2.1, 2.0, 1.9, 2.0]
  line "collapse" [2.2, 1.6, 0.8, 0.2, 0.1, 0.1]
```

residual_rms

Intuition. Layerwise activation scale.
Why it matters. Runaway scales often precede NaNs and can cause subtle behavior shifts even if loss

remains stable.
Good/bad patterns. Good: stable per-layer percentiles. Bad: one layer's RMS grows steadily; jumps after resume.
Failure modes. Normalization mismatch; unstable attention; outlier data.
Mitigations. Identify first diverging layer; correlate with max_attn_logit and overflow; reduce LR; quarantine outlier shards if localized. [36]

activation_outlier_rate

Intuition. Fraction of activations beyond a threshold (e.g., >kσ or >absolute bound).
Why it matters. Outlier rate often detects impending overflow earlier than RMS.
Good/bad patterns. Good: stable tail fraction. Bad: tail grows, especially in attention/MLP blocks.
Failure modes. Numeric instability; mixed precision sensitivity; data outliers.
Mitigations. Keep unsafe ops in FP32; tighten clipping; reduce LR; monitor correlations to overflow and attention metrics. [47]

logit_entropy_probe and topk_mass_probe

Intuition. Cheap output-distribution sharpness probes on a fixed input batch.
Why it matters. They can shift before loss changes, explaining "behavioral" regressions or degraded calibration.
Good/bad patterns. Good: smooth drift. Bad: step-change (often harness drift) or monotone collapse (overconfidence).
Failure modes. Unpinned decoding params; tokenizer/template changes; internal drift due to attention sharpness issues.
Mitigations. Pin and hash probe batch + decoding; if changed, invalidate comparisons; if real drift, check attention and LR. [47]

output_kl_to_prev

Intuition. KL divergence between model outputs at checkpoint t and $t-1$ on a frozen probe set.
Why it matters. Detects catastrophic behavior shifts even when scalar losses are stable.
Good/bad patterns. Good: small, smooth KL. Bad: spikes.
Failure modes. Corrupted checkpoint; resume mismatch; sudden data or optimizer change.
Mitigations. Rerun probe under deterministic mode; validate eval artifacts; rollback if KL spike aligns with val regression. [42]

moe_drop_rate

Intuition. Fraction of routed tokens dropped because an expert exceeds capacity.
Why it matters. Switch describes that if too many tokens are routed to an expert, computation for "dropped tokens" is skipped and the representation is passed via residual; it reports dropped tokens are typically <1% and emphasizes keeping drop rates low. [5]
Good/bad patterns. Good: ~0 (or within a tight budget). Bad: sustained nonzero; spikes in certain layers or sources.
Failure modes. Capacity factor too low; routing imbalance; domain skew; router collapse.

Mitigations. Increase capacity factor; improve balancing; inspect moe_util_entropy; if drops are source-local, rebalance data mix. [48]

```
xychart-beta
  title "MoE drop rate (illustrative)"
  x-axis "progress" [0, 1, 2, 3, 4, 5]
  y-axis "drop rate (%)" 0 --> 2
  line "healthy" [0.2, 0.1, 0.0, 0.0, 0.0, 0.0]
  line "bad"     [0.2, 0.6, 1.2, 1.4, 1.1, 1.3]
```

moe_util_entropy and moe_load_balance_loss

Intuition. How evenly experts are used, and whether balancing is enforced.
Why it matters. Switch states that using the auxiliary load balancing loss with a high enough coefficient ensured good load balancing; imbalance causes drop spikes or capacity underuse. [5]
Good/bad patterns. Good: stable utilization entropy; stable aux loss; low drops. Bad: entropy collapse; rising aux loss + rising drops.
Failure modes. Missing/buggy balancing loss; router collapse; data skew.
Mitigations. Verify the aux loss implementation matches the paper; sweep balancing coefficient on canaries; monitor drop and entropy jointly; adjust capacity factor. [5]

Data and distribution playbooks

source_mix_share

Intuition. Token share by source/domain.
Why it matters. Mixture drift is a top silent-regression cause; it changes what the model learns without necessarily moving loss sharply.
Good/bad patterns. Good: stable or planned curriculum. Bad: step-change drift after data refresh or filter update.
Failure modes. Source routing bug; filter yield drift; upstream data changes.
Mitigations. Alert on drift; quarantine new sources; run slice_val_loss for dipped benchmarks (e.g., code/math/long-context). [49]

language_share

Intuition. Token share by language.
Why it matters. Language-specific capabilities are highly sensitive to mix; drift can cause localized benchmark dips.
Failure modes. Lang-ID model drift; encoding failures.
Mitigations. Version lang-ID; add language slices; inspect bytes_per_token tail for encoding anomalies. [50]

filter_reject_rate

Intuition. Acceptance/rejection rates per filter.
Why it matters. Filters are part of the data contract; sudden yield changes imply upstream drift or filter

bugs.
Failure modes. Regex overmatching; new junk formats; filter parameter changes.
Mitigations. Sample rejects; rollback filter changes; correlate yield change with token_kl_drift and duplicates. [6]

bytes_per_token

Intuition. Simple proxy for junk/markup/binary ingestion.
Why it matters. Non-language junk wastes compute and harms generalization.
Failure modes. Broken decoding; HTML/script ingestion; mis-tokenization.
Mitigations. Clamp extremes; isolate sources; improve normalization. [28]

seq_len_p50_p95 and pad_fraction

Intuition. Length distribution and padding waste.
Why it matters. Length drift changes long-context exposure and throughput; padding waste directly reduces effective throughput and MFU. [14]
Mitigations. Enforce length bucketing/packing; gate changes via canary; track tokens_per_update jointly. [51]

token_kl_drift

Intuition. KL drift of token histogram.
Why it matters. Captures distribution shift that can produce benchmark dips without dramatic loss changes.
Mitigations. Localize by source/lang; run slice_val_loss. [52]

near_dup_rate and train_val_overlap

Intuition. Near-duplicate prevalence and leakage across splits.
Why it matters. The dedup paper finds many near-duplicates and long repetitive substrings; it reports over 1% of unprompted LM output is copied verbatim without dedup, dedup reduces memorization by 10×, reduces required steps, and reduces train–test overlap affecting >4% of validation sets. [6]
Mitigations. Run dedup at data-build time (MinHash/LSH); blacklist huge clusters; enforce overlap budgets as release gates; rerun memorization probes after data changes. [6]

doc_repeat_hotspots

Intuition. Detect giant repeated substrings/clusters.
Why it matters. The dedup paper gives an example of a 61-word sentence repeated >60,000 times, illustrating how repetition can dominate gradients. [28]
Mitigations. Identify and remove hotspot clusters; consider weighting or source blacklists. [28]

Compute and hardware playbooks

tokens_per_sec and step_time

Intuition. Throughput and latency.
Why it matters. MegaScale emphasizes maintaining high efficiency over long runs and using deep observability to diagnose failures/stragglers; throughput regressions are direct budget waste. [53]
Mitigations. Decompose step_time into input/compute/comm; correlate with dataloader_stall_frac and comm_time_frac; localize stragglers by rank. [29]

mfu

Intuition. Normalized training efficiency.
Why it matters. PaLM reports 57.8% hardware FLOPs utilization and provides MFU/HFU context; MegaScale reports 55.2% MFU at 12,288 GPUs and attributes gains to full-stack optimizations and diagnosis tooling. [4]
Good/bad patterns. Good: stable band. Bad: step-change drops after kernel/data changes; slow decay from increasing stalls.
Mitigations. Use MFU as a regression gate for low-level changes; investigate time breakdown; run canaries before rolling changes. [12]

```
xychart-beta
  title "MFU regression (illustrative)"
  x-axis "progress" [0, 1, 2, 3, 4, 5]
  y-axis "MFU (%)" 0 --> 80
  line "stable" [55, 56, 55, 55, 56, 55]
  line "drop"   [55, 54, 48, 46, 45, 46]
```

gpu_util and gpu_mem_used

Why it matters. Low utilization with high step time indicates stalls; memory creep predicts OOM storms. MegaScale reports diagnosing failures and stragglers “deep in the stack,” which often begins with these basic signals. [29]
Mitigations. Correlate with dataloader_stall_frac and comm_time_frac; enforce memory budgets; use bucketing/packing; consider activation checkpointing.

dataloader_stall_frac and comm_time_frac and compute_time_frac

Why it matters. These decompose step time and convert “throughput drop” into a diagnosable root cause class. MegaScale emphasizes overlapping compute/communication and tuning data pipelines. [29]
Mitigations. If input-bound: pre-tokenize, cache, prefetch. If comm-bound: overlap, tune buckets, isolate stragglers. If compute-bound: profile kernels.

straggler_ratio

Why it matters. A few slow ranks can dominate step time in distributed training; MegaScale explicitly targets mitigating stragglers and failures. [53]
Mitigations. Identify slow ranks; swap nodes; check network/thermal; correlate with error logs.

Safety and privacy playbooks

data_toxicity_rate and rtp_toxicity_prob

Intuition. Toxicity in training data sample and toxicity probability on a standard probe set.
Why it matters. RTP provides a standardized benchmark to measure toxic degeneration and reports dataset stats (100K prompts; 22K toxicity ≥ 0.5; prompt length 11.7±4.2 tokens). [7]
Mitigations. Pin decoding params; version the toxicity scorer; quarantine sources that spike toxicity; rerun on canaries.

pii_hit_rate

Why it matters. PII exposure is high impact and often correlates with memorization and duplication. [28]
Mitigations. Quarantine sources; rebuild datasets; use stricter filters; rerun memorization probes.

memorization_rate

Why it matters. Dedup work quantifies memorization and shows strong improvements with dedup. [6]
Mitigations. Stronger dedup; reduce repeats; monitor overlap budgets; use targeted probes for regurgitation.

Evaluation and validation playbooks

val_loss, val_ppl, train_val_gap

Why it matters. Detect overfitting and distribution mismatch; scaling-law work connects cross-entropy loss to scaling behavior and compute-optimality, making val loss a key global KPI. [21]
Mitigations. Run slice eval; inspect drift and duplicates; rollback mixture changes; check scaling-law residual.

slice_val_loss

Why it matters. Aggregate validation can hide slice regressions: long-context, code/math, languages. DeepSeek-V3.2 and MiMo-V2-Flash highlight long-context mechanisms, implying length-sliced eval is required when those mechanisms are used. [54]
Mitigations. Track per-slice deltas; rebalance mixtures; adjust context curriculum; test sparse/hybrid attention variants.

fixed_probe_loss

Why it matters. A “canary invariant” to detect pipeline drift. If it changes discontinuously, you likely changed tokenizer, masking, or preprocessing.

Mitigations. Store probe tokens and hashes in the reproducibility manifest; fail-fast on mismatch; rerun old checkpoint with new harness to isolate.

microbench_score and eval_artifact_hash

Why it matters. Silent regressions are often harness or prompt drift; L4 shows failures and diagnosis toil are real, so you need pinned artifacts to avoid chasing ghosts. [8]
Mitigations. Hash prompts/templates/tokenizer/decoding; invalidate comparisons on mismatch; rerun cross-product (old ckpt × new harness, new ckpt × old harness).

Observability and logging playbooks

nan_inf_events

Why it matters. Indicates numerical corruption; NVIDIA emphasizes skipping updates on overflow to avoid irreversibly damaging weights. [19]
Mitigations. SEV0 pause; snapshot last batches + logs; rollback to last-known-good checkpoint; debug replay.

checkpoint_ok, checkpoint_time, checkpoint_size

Why it matters. Checkpoint integrity is the prerequisite for safe rollback; time/size affect throughput. MegaScale emphasizes fault tolerance and diagnosis. [29]
Mitigations. Atomic writes; checksums; periodic restore tests; separate “last-known-good” checkpoint from “latest.”

restart_rate and error_event_rate

Why it matters. L4’s study finds hardware and user faults dominate and diagnosis relies on logs; structured error rates enable faster localization and better runbooks. [8]
Mitigations. Auto-capture diagnostic bundles; blacklist unstable nodes; gate changes via canaries; classify errors by stage and rank.

metrics_ingest_lag and heartbeat_missing

Why it matters. If observability lags, you are blind to fast failures; L4 relies on temporal/spatial/cross-job patterns which require timely telemetry. [8]
Mitigations. Cap label cardinality; prioritize SEV0 metrics; alert on missing time series; treat monitoring outages as incidents.

Silent benchmark regression diagnosis

When “training metrics look fine” but benchmarks dip, do not jump to architecture changes. Triage in a strict order:

1. **Evaluation harness drift** (artifact mismatch).
2. **Data drift** (mixture/filter/length/dup/leak).

3. **Internal behavior drift** (attention sharpness/entropy; MoE drops; probe KL).
4. **Architecture/capacity mismatch** (depth/width/heads/MoE/sparsity/context). [55]

```
flowchart TD
  R["Benchmark drop\n(loss stable,\nno NaNs)"] --> E{"Eval artifacts match?
\n(eval_artifact_hash)"}
  E -->|No| FixE["Pin artifacts\nrerun old ckpt\nunder new harness\nand vice-
versa"] --> Done1["Harness issue"]
  E -->|Yes| D{"Data drift?\n(mix/filters/KL/\ndup/overlap)"}
  D -->|Yes| FixD["Rollback/quarantine data\nslice eval\nre-canary"] -->
Done2["Data issue"]
  D -->|No| I{"Internal drift?\n(attn logits/entropy,\nMoE drops, probe KL)"}
  I -->|Yes| FixI["Stabilize\n(LR, QK-Clip,\nAMP policy)\nverify on canary"] -->
Done3["Dynamics issue"]
  I -->|No| A["Architecture mismatch\nrun controlled ablations\n(depth/width/
heads/\nMoE/sparsity)"]
```

Architecture troubleshooting guide

This section targets the question: “If a benchmark dips, what architecture axis should I test, and how?”

Architecture axes and their effect on benchmark classes

Use the mapping “benchmark class → required computation pattern → architecture axis.”

- **Long-context retrieval and reasoning:** depends on context length curriculum and attention mechanism (full vs sparse/hybrid). DeepSeek-V3.2 introduces DeepSeek Sparse Attention (DSA) to reduce attention complexity while preserving long-context performance; MiMo-V2-Flash uses hybrid local/global attention (128-token sliding window, 5:1 local:global ratio) and reports near 6× KV-cache reduction. [56]
- **MoE specialization (code/math/agentic traces):** depends on expert count, active experts, expert MLP width, and routing stability (drop rate, utilization entropy). Switch formalizes capacity and emphasizes keeping dropped tokens low (<1% typical in its experiments). [5]
- **Sharpness-sensitive stability:** depends on attention dynamics and LR; Apple and Kimi show how attention sharpeness proxies track instability. [38]

Recent SOTA architecture reports give concrete “knobs with measured tradeoffs”:

- Kimi K2 reduces attention heads to 64 (vs 128 in DeepSeek-V3) for inference efficiency; it reports that at 128k sequence length, increasing heads 64→128 increases inference FLOPs by 83% in a controlled comparison. [57]
- DeepSeek-V3 is a 671B total / 37B activated MoE model and reports training stability with no irrecoverable loss spikes or rollbacks. [58]
- GLM-5 reports scaling to 256 experts and reducing layer count to 80 to reduce expert-parallel communication overhead, yielding 744B total / 40B active. [59]

- MiMo-V2-Flash reports hybrid attention and long-context extension to 256k, with long-context retrieval success near 100% across 32K–256K in its evaluations. [60]

Targeted micro-experiments and heuristics

Use compute-budgeted canaries and change one axis at a time. For each, pin: tokenizer, data manifest, eval artifacts.

Depth vs width (fixed parameter or compute budget).
Test whether dips are due to insufficient compositional capacity (often depth-sensitive) or representational width. Kaplan's scaling-law paper notes that "other architectural details such as width or depth have minimal effects within a wide range," which is a warning: do not expect small depth/width perturbs to fix major dips unless you are at an extreme regime. [40]

Head count vs head dimension.
If dips are long-context or attention-related, test heads and head dim at fixed model width. Kimi K2 provides evidence that head count increases can be very expensive for long-context inference relative to validation gains, so treat this as a system-quality tradeoff, not just "more heads is better." [61]

MoE experts and capacity factor.
If dips are domain-specific in MoE, first test routing/capacity (drop rate, utilization entropy) before changing model width. Switch provides the formal capacity factor tradeoff and emphasizes low drop. [48]

Attention sparsity vs full attention.
If dips are long-context-only, evaluate a sparse/hybrid attention variant, because this is exactly what DeepSeek-V3.2 and MiMo-V2-Flash target. [54]

Operational practices and prioritized checklists

Dashboard layout (minimal but sufficient)

- **Per-step ICU:** train_loss_ema, grad_global_norm, update_to_weight, loss_scale + overflow_rate, steps_seen vs tokens_seen, max_attn_logit, sampled attn_entropy. [11]
- **Per-minute efficiency:** tokens_per_sec, step_time p50/p95, MFU, dataloader_stall_frac, comm_time_frac, straggler_ratio. [62]
- **Daily data health:** mixture shares, filter yields, length distribution, KL drift, duplicates/overlap. [28]
- **Per-checkpoint quality + safety:** val + slice eval, microbench, RTP toxicity, memorization probes, probe KL. [15]

Alerting rules and runbooks

SEV0 (pause/stop): nan_inf_events > 0; checkpoint_ok false; monitoring blackout; overflow storms for long windows. NVIDIA describes skipping weight updates on overflow to avoid irreversibly damaging weights; if this is sustained, stop and diagnose. [10]

SEV1 (contain quickly): repeated loss spikes; max_attn_logit runaway (>1000 is a red flag in Kimi); attention entropy collapse; MFU collapse; sustained moe_drop_rate >0. [23]

SEV2 (investigate): mixture drift; val/slice regressions; toxicity/memorization trend increases; rising duplicate clusters. [63]

Each alert should link to a runbook that specifies: evidence to capture (metrics window + last N batch IDs + top logs), safe containment action, and a reproducible "probe replay" step. L4's results justify investing in structured log event extraction and localization. [8]

Sampling strategies

- **Step-level:** cheap scalars (loss, LR, grads, AMP, throughput) + sampled max_attn_logit. [35]
- **Every 10–100 steps:** sampled attention entropy and activation tails. [25]
- **Checkpoint-level:** val + slice eval + microbench + RTP + memorization + probe KL. [15]

Synthetic probes and canary runs

- **Fixed probe batch:** immutable tokens for fixed_probe_loss, logit entropy, top-k mass, and KL-to-prev.
- **Canary runs:** 200–2,000 steps on fixed shards after any code/data/optimizer change. L4 shows failures are inevitable and diagnosis is costly; canaries reduce wasted compute. [8]

Checkpointing and reproducibility manifests

- **Checkpoint atomicity + restore tests:** required for safe rollback; store optimizer and scheduler states and AMP scaler states. [18]
- **Reproducibility manifest:** code commit + dependency lock; model config; tokenizer hash; data manifest (shards, filters, dedup version); eval artifact hashes and decoding params. This is the only way to interpret "silent regressions" rigorously. [13]

Nano vs medium/large prioritized checklists

Nano (resource constrained) must-have - Numeric safety: nan_inf_events, loss_scale, overflow_rate, skipped_updates. [19]
- Optimization: train_loss_ema, tokens_per_update, grad_global_norm, update_to_weight.
- Internal vitals: max_attn_logit (and optional sampled attn_entropy). [64]
- Efficiency: tokens_per_sec, step_time, dataloader_stall_frac.
- Data build-time: near_dup_rate, train_val_overlap. [6]
- Checkpoint-level: val_loss, slice_val_loss, RTP toxicity, memorization. [15]

Medium/large add-ons - MFU + time decomposition + per-rank straggler telemetry (MegaScale). [12]
- Automated log event extraction and localization (L4). [8]
- Full MoE routing telemetry and balancing tuning (Switch). [5]

Importance tables

By model scale

metric family	nano	medium	large
AMP + NaN/Inf safety	must	must	must [19]
Attention logits/entropy	must	must	must [64]
Dedup + overlap	must (build-time)	must	must [28]
Throughput/step-time	must	must	must [29]
MFU + stragglers	nice	should	must [14]
Log diagnosis tooling	should	must	must [8]
RTP + memorization probes	should	must	must [15]

By training phase

metric family	early	steady	late
AMP/NaN/Inf	must	must	must [19]
Attention stability vitals	must	must	should [64]
Data drift + dedup	should	must	must [28]
Val + slice eval	should	must	must [54]
Harness invariants	nice	should	must [32]
Safety probes	nice	should	must [30]

Limitations and open questions

This report provides a broad, practice-oriented metric set grounded in primary sources, but universal numeric thresholds are rare: most values depend on model size, tokenizer, data mix, precision (FP16/BF16/FP8), and architecture. We therefore emphasize trend shapes and coupling signals, and we cite concrete anchors only when primary sources provide them (e.g., QK-Clip τ=100 and >1000 attention-logit spikes; MFU/HFU values in PaLM and MegaScale; MoE drop "typically <1%" in Switch; RTP dataset composition and thresholds). [65] Architecture troubleshooting remains partly empirical: scaling-law evidence suggests many architectural details have limited effect over broad ranges, but recent SOTA models show attention efficiency (sparse/hybrid) and MoE routing choices can be first-order for long-context and specialization. The best open question is how to standardize *early-warning capability probes* (beyond loss) that are cheap, stable, and predictive of downstream dips across model families. [66]

---

[1] [29] [53] MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs
https://arxiv.org/abs/2402.15627?utm_source=chatgpt.com

[2] [24] [61] Kimi K2: Open Agentic Intelligence
https://arxiv.org/html/2507.20534v1?utm_source=chatgpt.com

3 25 37 38 46 Stabilizing Transformer Training by Preventing Attention ...
https://proceedings.mlr.press/v202/zhai23a/zhai23a.pdf?utm_source=chatgpt.com

4 14 PaLM: Scaling Language Modeling with Pathways
https://arxiv.org/pdf/2204.02311?utm_source=chatgpt.com

5 26 arXiv:2101.03961v3 [cs.LG] 16 Jun 2022
https://arxiv.org/pdf/2101.03961?utm_source=chatgpt.com

6 13 27 28 31 49 52 55 63 Deduplicating Training Data Makes Language Models Better
https://arxiv.org/abs/2107.06499?utm_source=chatgpt.com

7 15 30 arXiv:2009.11462v2 [cs.CL] 25 Sep 2020
https://arxiv.org/pdf/2009.11462?utm_source=chatgpt.com

8 17 32 42 L4: Diagnosing Large-scale LLM Training Failures via Automated Log Analysis
https://arxiv.org/abs/2503.20263?utm_source=chatgpt.com

9 10 11 18 19 33 35 41 47 Train With Mixed Precision - NVIDIA Docs
https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html?utm_source=chatgpt.com

12 51 62 MegaScale: Scaling Large Language Model Training to ...
https://www.usenix.org/system/files/nsdi24-jiang-ziheng.pdf?utm_source=chatgpt.com

16 23 34 36 45 57 64 65 Kimi K2: Open Agentic Intelligence
https://arxiv.org/pdf/2507.20534?utm_source=chatgpt.com

20 39 An Empirical Model of Large-Batch Training
https://arxiv.org/pdf/1812.06162?utm_source=chatgpt.com

21 40 43 66 Scaling Laws for Neural Language Models
https://arxiv.org/pdf/2001.08361?utm_source=chatgpt.com

22 44 Neglected Hessian component explains mysteries in ...
https://openreview.net/pdf?id=m6pVpdIN0y&utm_source=chatgpt.com

48 Switch Transformers: Scaling to Trillion Parameter Models ...
https://ar5iv.labs.arxiv.org/html/2101.03961?utm_source=chatgpt.com

50 MiMo-V2-Flash Technical Report
https://arxiv.org/abs/2601.02780?utm_source=chatgpt.com

54 56 DeepSeek-V3.2: Pushing the Frontier of Open Large ...
https://arxiv.org/pdf/2512.02556?utm_source=chatgpt.com

58 DeepSeek-V3 Technical Report
https://arxiv.org/pdf/2412.19437?utm_source=chatgpt.com

59 GLM-5: from Vibe Coding to Agentic Engineering
https://arxiv.org/html/2602.15763?utm_source=chatgpt.com

60 MiMo-V2-Flash Technical Report
https://www.arxiv.org/pdf/2601.02780?utm_source=chatgpt.com