

New terms.

Clear meanings.

Each definition has a plain explanation, a technical description and an example. The + terms throughout the guided brief open these definitions in place.

Attention

A learned calculation that mixes information from relevant positions in the available context.

Technically: Queries compare with keys; normalized scores combine values. Attention weights are computational coefficients, not guaranteed explanations of the model's reasoning.

Example: When generating a pronoun, a layer may draw on earlier names and nouns, while other layers perform different context interactions.

Autoregressive generation

Generating a sequence by extending it one token at a time.

Technically: Each next-token distribution is conditioned on earlier input and generated tokens. Ordinary decoding feeds the selected token back into the model.

Example: A report assistant builds its response incrementally; a longer response usually requires more decoding steps.

Attention head

One parallel channel for mixing context within an attention layer.

Technically: Each query head uses a projected subspace. Head outputs are combined and projected back into the model's hidden representation.

Example: Several heads can process different learned relationships at once; they are not labeled human experts such as “finance” or “legal”.

Backpropagation

The calculation that carries the loss signal backward through the model.

Technically: The chain rule combines derivatives through successive operations to compute gradients of the loss with respect to trainable parameters.

Example: An error on the final predicted token can influence earlier attention and feed-forward weights through their contributions to that prediction.

Clear meanings.

Batching

Processing work from several examples or requests together.

Technically: Batches improve device utilization by combining tensor operations. Dynamic or continuous batching can change membership as requests begin and finish.

Example: A server can group incoming requests to raise total output rate, while queueing can increase some users' waiting time.

Causal mask

A rule that blocks access to tokens that come later in a sequence.

Technically: At position t , future attention scores are excluded, conventionally by adding $-\infty$ before softmax. This prevents next-token training from reading future targets.

Example: When learning to predict the next word after “The cat”, the model cannot peek at the rest of the sentence.

Benchmark / contamination

A standardized evaluation; contamination occurs when evaluation content leaks into training or selection.

Technically: Scores depend on prompts, tools, sampling, scoring and dataset exposure. Comparisons require compatible conditions and a relevant workload.

Example: A high public coding score is useful evidence, but an internal code-test evaluation better estimates success on your repository.

Checkpoint

A saved version of the model's learned settings.

Technically: A deployable release usually combines parameter files with architecture configuration, tokenizer, generation settings and sometimes adapters. Exact versions matter.

Example: Pin a checkpoint hash so a procurement evaluation and the production service use the same model version.

Clear meanings.

Concurrency

How many requests are in progress at the same time.

Technically: A request can be queued, prefilling or decoding. State the measurement boundary: active GPU sequences, in-flight API requests and connected users are different counts.

Example: Twenty staff members can share a service; the capacity test must check how many requests overlap and whether response times stay acceptable.

Cost per accepted task

The total workflow cost divided by the number of outcomes that meet your requirements.

Technically: Include serving, retrieval, retries, human review, idle capacity and operating work. Define acceptance criteria before comparing solutions.

Example: A cheaper token rate can still cost more per approved report if it produces more errors and review work.

Context window

The maximum amount of tokenized context supported in a request under the model and serving setup.

Technically: The budget includes prompt material and generated continuation, and can include representations for images. Runtime limits and usable quality can be lower than an advertised maximum.

Example: A chatbot history, system instructions, retrieved pages and the answer all consume context; a large limit does not guarantee accurate recall.

Cross-entropy / negative log likelihood

A score that becomes smaller when the model gives more probability to the observed answer.

Technically: With a one-hot target, $L = -\ln p(\text{correct token})$. A sequence objective sums or averages this quantity over selected tokens. The natural logarithm gives units called nats. Here $\ln$ is the natural logarithm, with base e (about 2.718). At probability 1 the score is 0; as the target probability approaches 0, the score grows. It is an objective value, rather than a percentage accuracy.

Example: Assigning very low probability to the observed “cat” creates a larger error signal than assigning it high probability.

Clear meanings.

Decode

The stage that extends the answer using the prompt and generated history.

Technically: Ordinary autoregressive decoding computes successive next-token distributions and reuses cached context. Optimizations can alter scheduling or verify several candidate tokens at once.

Example: Generating a 1,000-token commentary usually takes more decoding work than generating a 50-token classification.

Dense model

A model whose regular feed-forward network participates for every token.

Technically: Dense contrasts with sparse expert routing. It does not mean every stored parameter is read for every token; embedding lookups and conditional paths can differ.

Example: Dense models often have simpler memory placement than very large expert models, though size and runtime still determine performance.

Decoder

The transformer component that uses available earlier context to predict the next token.

Technically: A decoder stack applies causal self-attention and feed-forward transformations, then projects hidden states to vocabulary logits. “Decoder” here is an architecture term.

Example: The decoder does not read a hidden answer key; it computes the next-token distribution from the prompt and previously generated text.

Direct preference optimization (DPO)

Training directly from pairs of preferred and less-preferred responses.

Technically: DPO uses a preference loss based on policy and reference log-probability ratios. The common formulation avoids a separately trained reward model and an online RL rollout loop.

Example: Provide two answers to the same analysis request and a reviewed preference; the method increases the relative likelihood of the preferred answer.

Clear meanings.

Distillation / teacher and student

Training one model to imitate useful behavior from another model.

Technically: Distillation can use teacher probability distributions or generated examples. A student keeps its own architecture and does not automatically inherit the teacher's abilities.

Example: A smaller model can learn from reviewed examples produced by a larger one and serve a narrow task at lower cost.

Embedding

A numeric representation of a token or another item that the model can calculate with.

Technically: A token embedding maps an ID to a vector. Later layers produce context-dependent hidden states; retrieval embeddings are a related but separate application.

Example: The word “bank” starts from a token representation, then surrounding words help layers distinguish a lender from a river bank.

Early fusion

Combining representations of different input types inside a shared model-processing path.

Technically: Text and visual representations interact through the decoder rather than only through a final combination of independent predictions. Implementation details vary.

Example: The model can use the question text while interpreting features from an attached diagram.

Expert router

The learned component that chooses expert paths for a token.

Technically: Routing scores select a limited number of experts. Gating, grouping, balancing and dispatch rules are implementation-specific.

Example: Two tokens in the same customer request can be routed differently; the system is not assigning the whole request to a separate complete model.

Clear meanings.

Feed-forward network (FFN / MLP)

The learned transformation applied to each token representation after context has been mixed.

Technically: An FFN expands the hidden vector, applies nonlinear operations and projects back. Within that sublayer, positions are processed independently; previous attention already mixed context.

Example: An invoice token's context-enriched representation is transformed into features that later layers can use for extraction.

Generalization / overfitting

Generalization is doing well on new cases; overfitting is learning details that help training cases but fail to transfer.

Technically: Training loss and held-out performance measure different things. Distribution changes, leakage and repeated evaluation tuning can make apparent improvements misleading.

Example: An invoice extractor must work on unseen suppliers and layouts, not only the samples used to fine-tune it.

Fine-tuning

Additional training on a narrower set of examples or goals.

Technically: Fine-tuning updates all parameters or a selected subset. The training objective and data quality determine the behavior learned.

Example: Train on approved extraction examples to improve consistent JSON output, then evaluate on invoices excluded from training.

Global attention

Attention that can access the full permitted earlier context in a layer.

Technically: In a causal decoder, global still excludes future positions. Full-context attention has a different memory and compute cost from local windows.

Example: Periodic global layers can connect clauses far apart in a long document.

Clear meanings.

GPU memory / VRAM

The device memory that holds weights and working state during model execution.

Technically: Serving requires weights, caches, activations, temporary workspaces and allocator overhead. Multi-device partitioning changes where that memory is needed.

Example: A nominal 16 GB of weights does not imply that a 16 GB GPU can run the model with long inputs and many requests.

Group relative policy optimization (GRPO)

An RL method that compares several generated responses to the same prompt.

Technically: A common form normalizes rewards within a group to estimate relative advantages, then applies a clipped policy update with a reference-policy penalty. Exact variants differ.

Example: If one of four candidate solutions passes a verifier, its relative reward can guide training; equally scored candidates provide little relative signal.

Gradient

Numbers showing how a small change to each parameter would change the loss.

Technically: A gradient is a vector of partial derivatives. It provides a local direction for adjustment, rather than a guarantee of the best possible model.

Example: Training computes whether slightly increasing or decreasing a particular weight would reduce the measured error on the current batch.

Grouped-query attention (GQA)

Several query heads share the same key and value heads to reduce stored context state.

Technically: If 32 query heads share 8 K/V heads, groups of 4 query heads reuse each K/V head. This reduces cache size relative to 32 separate K/V heads.

Example: A serving team can support longer prompts or more simultaneous requests with less K/V memory, subject to the rest of the architecture.

Clear meanings.

Hallucination / unsupported output

Generated content that is false, fabricated or not supported by the supplied evidence.

Technically: A plausible next-token sequence need not correspond to a true claim. Fluent wording, high token probability and confident tone do not establish factual accuracy.

Example: The model may invent an invoice total or citation; compare consequential outputs with authoritative records.

Holdout / evaluation set

Examples reserved for judging performance instead of teaching or selecting the model.

Technically: Keep development cases separate from final evaluation cases. Reusing a holdout repeatedly for decisions can turn it into a development set.

Example: Reserve a later month of documents and difficult exceptions to test how the selected workflow handles unseen work.

Hidden state / activation

The temporary numbers produced while processing an input.

Technically: An activation is computed from the current input and weights. Hidden states change from layer to layer; they are not the stored model parameters.

Example: Two customers' requests use the same weights but create different temporary activations and caches.

Inference

Running a trained model to produce an output for a new input.

Technically: Ordinary inference performs forward computation and token selection with fixed learned parameters. It may create temporary caches, but does not update weights.

Example: A finance assistant summarizes today's report. The report influences that response through the prompt, without automatically becoming a permanent learned fact.

Clear meanings.

Key/value cache (K/V cache)

Temporary saved context calculations that let generation avoid recomputing earlier keys and values.

Technically: A cache stores per-layer state for processed positions. Its size depends on sequence lengths, K/V design, data type and request count. MLA uses a different cache representation.

Example: Keeping a long conversation active can use substantial GPU memory even when the model's weight memory is unchanged.

Latency

The time a user or application waits for an operation.

Technically: End-to-end latency includes queueing, prompt processing, generation and application overhead. Specify start and end points when comparing measurements.

Example: An extraction task that finishes in 3 seconds has different usability from a chat answer that begins in 0.5 seconds but takes 10 seconds to finish.

Large language model (LLM)

A model trained to work with language by predicting and generating pieces of text.

Technically: A language model estimates a probability distribution over token sequences. The models in this brief use autoregressive transformer decoders.

Example: An analyst gives a model a revenue table and asks it to draft a commentary; the draft still needs checking against the table.

Local / sliding-window attention

Attention restricted to a nearby span of tokens.

Technically: A window limits the positions available to a query in selected layers, reducing attention work and potentially cache retention for those layers.

Example: A layer can process nearby contract text efficiently while other layers provide broader connections.

Clear meanings.

Logits

Raw scores for candidate next tokens before they become probabilities.

Technically: The vocabulary projection maps a final hidden state to one score per vocabulary token. Softmax then normalizes these scores.

Example: Scores of 2, 1 and 0 for three candidate tokens do not mean probabilities of 200%, 100% and 0%.

Low-rank adaptation (LoRA)

Fine-tuning a small additional set of parameters while keeping the main model weights fixed.

Technically: For selected matrices, an update is represented as $\Delta W = (\alpha/r)BA$ with rank r much smaller than the matrix dimensions. Adapters can be loaded or sometimes merged.

Example: A team can train a compact adapter for a consistent report style without retraining every base parameter; dataset and evaluation quality still matter.

Loss / numerical error

A number used during training to measure how poorly a prediction fits the training target.

Technically: For an observed target token, cross-entropy loss is $-\ln p(\text{target})$. Larger loss means the model assigned less probability to the observed answer.

Example: If the next training token is “cat”, probability 0.10 gives loss 2.303; probability 0.80 gives loss 0.223. Nobody is being punished.

Machine learning (ML)

A way to build software by adjusting numeric settings from examples, rather than writing every decision as a rule.

Technically: Training selects model parameters that improve a stated objective on data. Learning a useful pattern does not guarantee it works on every new case.

Example: Instead of coding every way an invoice describes a date, train a model on labeled invoices and test it on unseen suppliers.

Clear meanings.

Mixture of experts (MoE)

A layer with several alternative feed-forward networks, where a router chooses which ones process each token.

Technically: Sparse MoE activates a subset of routed FFN experts, sometimes with shared experts. Attention and other components also perform work.

Example: A model can have many stored parameters while using a smaller subset per token; the full checkpoint still needs storage and placement.

Multi-head latent attention (MLA)

An attention design that stores a compressed representation of context instead of a full conventional set of keys and values.

Technically: MLA uses learned low-rank K/V representations, with separate positional handling. Efficient runtimes exploit this representation to reduce cache memory; it is not ordinary GQA.

Example: For long-context serving, a smaller context cache can free memory for requests, while model weights and communication remain separate costs.

Model card / license

A model card describes a release; a license states permissions and obligations for its use.

Technically: Cards summarize intended use, architecture, evaluation and disclosures. They may omit a reconstructable dataset or full recipe. Read the exact checkpoint's terms.

Example: Before selecting a model, record its card, weight license and acceptable-use requirements alongside technical evaluation results.

Multimodal / vision encoder

Working with more than one input type, such as images and text; a vision encoder converts images into numerical features.

Technically: Visual features are projected into representations used by the language model. Supported inputs and outputs depend on the exact model release.

Example: An assistant can read a supplied chart image and answer in text; image understanding does not imply image generation.

Clear meanings.

Normalization / RMSNorm

A calculation that controls the scale of a vector during model processing.

Technically: RMSNorm rescales a vector using its root mean square and learned gains. Placement and additional normalization vary by model.

Example: Keeping activations at manageable scales helps a deep stack train and compute reliably; it is not cleaning the training dataset.

Optimizer / learning rate

The update rule that turns gradients into changes to learned settings; the learning rate controls the size of those changes.

Technically: Simple gradient descent uses $\theta \leftarrow \theta - \eta \nabla L$. Optimizers such as Adam also track running gradient statistics. η is the learning rate.

Example: Too-large updates can destabilize fine-tuning; too-small updates can make useful adaptation slow.

Open-weight / closed-weight

Open-weight releases let users obtain model parameters under a license; closed-weight services retain those parameters.

Technically: Weight access is separate from source-code access, training-data disclosure, hosting arrangements and license permissions. Open-weight does not automatically mean unrestricted or reproducible training.

Example: Your team can self-host released weights or buy a managed endpoint using them; either option still has terms and operating costs.

p95 latency / percentile

A response-time threshold met by 95% of measured requests.

Technically: The 95th percentile describes the slow tail of a specific workload sample. It is neither the average nor a guarantee that every future request meets it.

Example: If p95 is 8 seconds, roughly 5% of sampled requests took longer; include failures and timeouts when evaluating the workflow.

Clear meanings.

Pan and scan

Processing additional image crops so small details can survive image resizing.

Technically: A vision preprocessing path supplies a base image and selected crops or patches. Additional visual inputs can increase processing and context cost.

Example: Small labels in a wide business dashboard may become easier to inspect, but more crops can slow the response.

Precision / bits per value

How much numeric detail is used to represent each stored or computed number.

Technically: FP32, BF16, FP16, FP8 and integer formats differ in range and precision. A 4-bit weight representation typically needs additional metadata.

Example: An 8-billion-parameter model needs about 16 decimal GB for bare 16-bit weights, before cache and runtime overhead.

Post-training

Training after the broad initial stage to shape how the model behaves.

Technically: This can include supervised examples, preference optimization, reinforcement learning and distillation. Different releases use different sequences of methods.

Example: A team adapts a broad model to give structured answers, cite provided evidence and handle task-specific instructions.

Prefill

The stage that processes the prompt before generated tokens begin.

Technically: Prompt positions can be computed in parallel within a causal forward pass, populating attention cache state. Long prompts add work before the first output token.

Example: Pasting a 100-page report can increase the wait before the first word appears.

Clear meanings.

Pretraining

The large initial learning stage that builds broad language and pattern recognition ability.

Technically: For autoregressive text models, a common objective is predicting observed next tokens across large datasets. Multimodal training may add image-text objectives or inputs.

Example: A pretrained base model can continue text; further training can teach it to respond in a useful assistant format.

Queries, keys and values (Q / K / V)

Three learned views of a token: what it seeks, what it can match, and what information it contributes.

Technically: Typically $Q = XW_Q$, $K = XW_K$ and $V = XW_V$. Dot products between Q and K determine coefficients used to combine V.

Example: For a question about a delivery date, the query can match contextual representations around relevant dates and mix their values.

Quantization

Storing or computing some model numbers with fewer bits.

Technically: Quantization maps values to a lower-precision representation, often with group scales and sometimes zero points. Weight, activation and cache quantization are separate choices.

Example: A lower-bit checkpoint can fit on fewer GPUs; validate extraction accuracy, runtime support and real speed with your workload.

Reinforcement learning (RL)

Training that adjusts behavior using scored outcomes from generated actions or responses.

Technically: A policy generates outputs; a reward or feedback signal guides updates to increase expected reward under the training setup. Language-model rewards are usually computed after generation.

Example: Reward generated code when it passes a test suite, while recognizing that incomplete tests can miss important defects.

Clear meanings.

Reinforcement learning from human feedback (RLHF)

Using human judgments to train a model toward preferred responses.

Technically: A classic RLHF pipeline trains a reward model from ranked answers, then optimizes the language model with an RL algorithm, often constrained by a reference policy.

Example: Reviewers can favor answers that follow instructions and explain uncertainty; their preferences do not automatically establish factual truth.

Residual connection

A path that adds a layer's transformation back to its input.

Technically: A residual update has the form $x \leftarrow x + f(x)$. It helps preserve information and supports optimization through deep stacks.

Example: Each block can refine the representation while retaining an incoming signal, rather than replacing every feature from scratch.

Reproducibility / version pinning

Keeping enough details to repeat an evaluation or deployment meaningfully.

Technically: Pin checkpoint, tokenizer, prompt template, runtime, precision and decoding settings. Seeds help with sampling, but hardware and kernels can still affect exact outputs.

Example: A quality regression investigation needs the tested settings and source versions, rather than only the model family name.

Retrieval / vector search

Finding records or passages likely to help answer a request.

Technically: Vector search ranks embeddings by a similarity measure; keyword and hybrid search are alternatives. Similarity is not a proof that a passage answers the question.

Example: Search for the applicable contract clause, then check the retrieved clause's date, scope and relevance before generating an answer.

Clear meanings.

Retrieval-augmented generation (RAG)

Finding relevant external material and supplying it to the model as context for an answer.

Technically: A retrieval pipeline selects records or document passages, often with embeddings and ranking. The generator uses the retrieved text without updating base weights.

Example: Retrieve the current expense policy and ask the model to answer using those passages and cite their source.

Rotary positional embeddings (RoPE)

A way to make attention sensitive to where tokens appear in a sequence.

Technically: RoPE rotates pairs of query/key coordinates using position-dependent angles. It changes their comparisons rather than adding a position vector to each token embedding.

Example: “Buyer pays seller” differs from “Seller pays buyer”; token order needs to influence the calculation.

Reward / verifiable reward

A numerical score used to prefer some generated outcomes during training.

Technically: Rewards can come from human judgments, learned reward models or executable checks. A verifiable reward checks a defined property, not every aspect of correctness.

Example: A math checker can reward an exact final answer; that score alone does not validate every explanation step or its use in another domain.

Sampling / greedy decoding

The rule that selects the next token from the predicted distribution.

Technically: Greedy decoding chooses the highest-probability token. Sampling draws from a distribution, usually modified by generation controls. Neither selection rule updates weights.

Example: For repeatable extraction, control decoding settings; for drafting alternatives, sampling can produce varied outputs.

Clear meanings.

Shared expert

An expert path used alongside whichever routed experts are selected.

Technically: A shared FFN processes tokens without top-k selection. Its result is combined with routed computation according to the model's implementation.

Example: Shared computation can provide a common path across tokens while routed paths offer additional capacity.

Supervised fine-tuning (SFT)

Teaching the model with examples of a prompt and a desired response.

Technically: The model commonly minimizes token-level negative log likelihood on target response tokens; prompt tokens may be masked from the training loss.

Example: Provide examples of “summarize this incident” followed by a reviewed summary with the required fields.

Softmax

A calculation that turns a list of scores into positive probabilities that sum to one.

Technically: $p_i = \exp(z_i) / \sum_j \exp(z_j)$. Implementations subtract the maximum logit first to improve numerical stability.

Example: Logits [2, 1, 0] become approximately [0.665, 0.245, 0.090]; a token can be the favorite without being certain.

SwiGLU / gated feed-forward

A feed-forward design in which one learned branch controls another branch's contribution.

Technically: SwiGLU commonly computes $W_{out}[\text{SiLU}(W_{gate} x) \odot (W_{up} x)]$, where $\text{SiLU}(a) = a \cdot \text{sigmoid}(a)$ and $\odot$ is elementwise multiplication. Other models can use other activations.

Example: The gate adjusts feature contributions numerically; it does not evaluate a business rule or open an access-control gate.

Clear meanings.

Temperature

A setting that changes how concentrated the token-selection probabilities are.

Technically: For positive T , $\text{softmax}(z/T)$ sharpens as T decreases and flattens as T increases. Many runtimes treat $T = 0$ as greedy selection.

Example: Higher temperature can vary wording, but it does not increase factual reliability or teach the model new information.

Throughput

How much work a system completes in a period of time.

Technically: Common serving metrics are output tokens per second across all requests and completed requests per second. Input and output token rates should be reported separately.

Example: A batch summarization service may prioritize 1,000 completed documents per hour over instant responses for one document.

Tensor

An organized array of numbers, such as a vector, matrix, or a stack of matrices.

Technically: Tensor dimensions describe the shape of model weights and intermediate activations. Hardware executes operations on these arrays.

Example: An embedding table can be a matrix with one row per vocabulary token and thousands of columns per row.

Time per output token (TPOT) / inter-token latency

How quickly additional tokens arrive after generation starts.

Technically: TPOT commonly averages elapsed generation time after the first token over the remaining output tokens. Inter-token latency measures individual gaps; report the convention.

Example: A service that emits a token every 25 ms delivers about 40 tokens per second to that request after startup, under that measurement.

Clear meanings.

Time to first token (TTFT)

How long a request waits before its first generated token arrives.

Technically: TTFT generally includes queueing and prefill plus the initial output step; network and application boundaries affect the metric.

Example: A streaming assistant can feel responsive with low TTFT even when a long answer takes longer to complete.

Tokenizer

The component that converts text into token IDs and converts generated IDs back into text.

Technically: Tokenization applies a fixed vocabulary and encoding rules. Different model families can produce different token counts for the same text.

Example: Use the model's tokenizer to estimate input cost; a rough words-to-tokens conversion is not reliable for every language or codebase.

Token

One piece of input or output text that the model handles at a time.

Technically: A token may represent a whole word, part of a word, punctuation, whitespace, or a code fragment. Token boundaries depend on the tokenizer.

Example: A long contract can exceed a model's input limit even when its word count appears small; measure its tokens with the actual tokenizer.

Top-p / nucleus sampling

A sampling rule that limits choices to a probability-ranked set of candidate tokens.

Technically: Keep the smallest high-probability prefix whose cumulative probability reaches p , then renormalize and sample. Implementations can combine it with other filters.

Example: With probabilities 0.60, 0.25 and 0.15 and top-p 0.80, the first two candidates remain because together they reach 0.85.

Clear meanings.

Total parameters / active parameters

Total counts describe what the model stores; active counts describe the subset participating in a token's computation.

Technically: For sparse models, active counts include selected experts and shared components under the publisher's counting convention. They are not direct measurements of speed or RAM demand.

Example: A model advertised with a small active count can still require a very large checkpoint and several devices.

Transformer

A neural network design that alternates context mixing with learned transformations of each position.

Technically: A decoder block usually contains attention, a feed-forward sublayer, normalization and residual connections. Many blocks are stacked.

Example: Attention connects "it" to an earlier product description; the feed-forward path transforms the resulting representation for the next layer.

Training

The process that changes learned settings using examples and a numerical objective.

Technically: A forward pass produces predictions, a loss measures error, backpropagation computes gradients, and an optimizer updates weights or adapters.

Example: Fine-tuning on approved support replies is training; merely asking the deployed assistant a support question is inference.

Vocabulary

The set of token pieces the model can select from.

Technically: The output head usually produces one logit for each vocabulary token. A large vocabulary is not a list of all facts the model knows.

Example: A rare product name may be represented by several familiar pieces even if there is no single token for that name.

Clear meanings.

Weights / parameters

The learned numbers that determine how the model transforms an input into predictions.

Technically: Weights are numeric tensors, often matrices. A parameter count measures learned scalar values, not documents, facts, or lines of source code.

Example: Downloading a checkpoint gives your team the learned settings needed to run the model; it does not supply the original training corpus.

What “penalizing low probability” actually means

“Penalize” is shorthand for increasing a numerical error score. There is no punishment of a person, and the model does not feel approval or disapproval.

Imagine the training text ends “The pet is a cat.” At the position before “cat”, the recorded next token is the training target. The model produces a distribution over its vocabulary. Cross-entropy reads the probability assigned to that observed target and converts it into a loss.

If the probability is 0.10, the loss is $-\ln(0.10) = 2.303$. If it is 0.80, the loss is $-\ln(0.80) = 0.223$. Giving the observed token more probability therefore reduces the error score. For a whole sequence or batch, training sums or averages these scores over the tokens included in the objective.

The target is what appears in the example, not necessarily the only sensible next word. Another sentence could use “dog”. The model learns a distribution across many examples, including their contradictions and quality limits. A lower training loss does not by itself prove factual correctness, helpfulness or generalization.

Backpropagation calculates how changing trainable parameters would change the loss. The optimizer makes updates intended to reduce the training objective over time. During ordinary inference, that update step is absent: the model computes and selects tokens using fixed weights.

Reading the notation: θ denotes the trainable weights; x_t is the observed token at position t ; the earlier x values are its prefix. $\ln$ is the natural logarithm, with base e (about 2.718). A probability of 1 gives zero loss. Using this logarithm measures the score in nats; this is a unit of information, rather than a percent error.

$$L_t(\theta) = -\ln p_{\theta}(x_t \mid x_1 \dots x_{t-1})$$

L_{batch} = mean of the selected token losses

Worked example. Observed target: cat. Prediction A: $p(\text{cat}) = 0.10 \rightarrow$ loss 2.303 nats. Prediction B: $p(\text{cat}) = 0.80 \rightarrow$ loss 0.223 nats. All values are a teaching example.

From scores to probabilities, then to an update

A model predicts scores first. Softmax turns those scores into a distribution, and the training target determines the error signal.

For a toy vocabulary [cat, dog, bird], let the logits be [2, 1, 0]. Exponentiating and normalizing gives probabilities approximately [0.6652, 0.2447, 0.0900]. If the target is cat, its loss is $-\ln(0.6652) \approx 0.4076$.

For softmax with this target, the derivative with respect to each logit is $p_i - y_i$, where y is [1, 0, 0]. The derivative is approximately [-0.3348, +0.2447, +0.0900]. Reducing loss therefore pushes toward a higher cat score and lower competing scores in this local calculation.

Real training updates weights, not a permanent table of scores for this one sentence. The chain rule connects these logit derivatives to millions or billions of upstream parameters. Updates from a batch combine many examples, and a particular example's loss may rise even when the overall training objective improves.

For numerical stability, subtract the largest logit before exponentiating; this leaves the probabilities unchanged. During inference, greedy decoding would choose cat in this example. Sampling could choose another token. Neither choice performs the gradient update.

$$\begin{aligned} p_i &= \exp(z_i - \max_j z_j) / \sum_j \exp(z_j - \max_j z_j) \\ \partial L / \partial z_i &= p_i - y_i \\ \theta_{\text{new}} &= \theta_{\text{old}} - \eta \nabla \theta L \end{aligned}$$

Worked example. A still smaller toy optimizes two logits directly: [0, 0] gives $p(\text{cat}) = 0.50$ and loss 0.6931. One gradient step with $\eta = 0.1$ makes the logits [0.05, -0.05], giving $p(\text{cat}) \approx 0.5250$ and loss ≈ 0.6444 . This isolates the update idea; a real optimizer changes model parameters.

A worked attention calculation

Attention turns a query's comparisons with available keys into a weighted mixture of their values.

Use a single toy head with two-dimensional vectors. Let $q = [1, 0]$, $k_1 = [1, 0]$ and $k_2 = [0, 1]$. Their dot products are 1 and 0. Divide by $\sqrt{2}$, giving scores $[0.7071, 0]$. Softmax produces coefficients approximately $[0.6698, 0.3302]$.

With values $v_1 = [2, 0]$ and $v_2 = [0, 2]$, the attention result is $0.6698 \cdot v_1 + 0.3302 \cdot v_2 \approx [1.3395, 0.6605]$. The query and keys decide the mixture; the values supply what is mixed. In a real model, learned matrix projections create Q, K and V from hidden states.

A causal mask excludes future positions before softmax. If a third key belongs to a later token, it receives a masked score of $-\infty$, so its coefficient is zero regardless of how well it would otherwise match. This allows next-token training on many positions at once without revealing future targets.

Many heads perform such calculations in parallel, and the resulting representations pass through an output projection and residual path. Some architectures normalize Q/K or alter score scaling, so this is the standard teaching equation rather than a literal specification for every release. Attention coefficients alone are not a faithful narrative of why a model answered.

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k} + \text{causal mask})V$$

Worked example. Toy $q = [1, 0]$; keys = $[[1, 0], [0, 1]]$; values = $[[2, 0], [0, 2]]$. Result $\approx [1.3395, 0.6605]$. These vectors are invented for explanation.

What a decoder block does, beyond attention

Attention connects positions. The feed-forward path transforms each resulting representation; residual paths and normalization keep the stack usable.

A common pre-normalized block first normalizes the incoming state, applies causal attention, then adds the result back to the incoming state. A second normalized branch applies a feed-forward network and adds its result. Stacking blocks repeatedly mixes and transforms information. Final normalization and a vocabulary projection produce logits.

A feed-forward network is not another pass across the sentence. Within that sublayer, each position's vector goes through learned projections and nonlinear operations. Earlier attention has already made the vector context-dependent. A dense block uses its regular FFN; an MoE block can substitute selected expert FFNs.

SwiGLU is one gated FFN pattern. One branch passes through SiLU, a smooth nonlinear function; it then multiplies another learned branch element by element before projection back to the hidden dimension. This gate is arithmetic, not a hand-authored decision rule. Other activation designs exist, so “gated FFN” should not be treated as a promise that every model uses identical SwiGLU.

The saved Llama 4 implementation uses SiLU-gated feed-forward paths. Gemma's inspected configurations document additional normalization and Q/K normalization. These differences matter when matching a checkpoint to an implementation; the simple block diagram remains a teaching schematic.

```
h1 = h + Attention(Norm(h))
h2 = h1 + FFN(Norm(h1))
SwiGLU(x) = Wout[SiLU(Wgate x) ⊙ Wup x]
SiLU(a) = a · sigmoid(a)
```

Worked example. If one projected gate value is 1 and the other branch value is 2, their gated product is $\text{SiLU}(1) \cdot 2 \approx 0.7311 \cdot 2 = 1.4621$ before the output projection. This scalar example illustrates one element, not a full model layer.

Release evidence [2, 5]

GQA, RoPE and long context solve different problems

GQA reduces how many K/V representations are stored. RoPE adds position sensitivity to attention comparisons. Context limits and context quality are separate questions.

In conventional multi-head attention, query heads can each have their own key and value head. In grouped-query attention, several query heads share one K/V head. With 32 query heads and 8 K/V heads, each group of 4 query heads uses shared K/V state. The model still has 32 query channels, but the conventional cache has one quarter as many K/V-head entries as a 32-K/V-head alternative with otherwise identical dimensions.

RoPE rotates coordinate pairs in queries and keys according to position. For a pair $[a, b]$, a rotation by angle ϕ produces $[a \cos\phi - b \sin\phi, a \sin\phi + b \cos\phi]$. Using related position-dependent rotations makes query/key dot products sensitive to relative positions. RoPE is not a learned document database and is not an extra vector simply added to the embedding.

Llama 3.1's card states that all sizes use GQA. Gemma 3 configurations also set fewer K/V heads than query heads and use separate local/global rotary settings. DeepSeek's MLA addresses cache efficiency through another representation.

Increasing an advertised context limit does not guarantee that the model will find a small fact buried anywhere in that context. Evaluate retrieval accuracy, prompt processing cost, memory and long-input behavior using realistic tasks.

Query group size = $n_{\text{query}} / n_{\text{KV}}$

RoPE pair rotation = $[[\cos\phi, -\sin\phi], [\sin\phi, \cos\phi]] \cdot [a, b]^T$

Worked example. For $[a, b] = [1, 0]$ and $\phi = 90^\circ$, the rotated pair is $[0, 1]$. Real RoPE uses many coordinate pairs and frequencies; this is only a geometric illustration.

Release evidence [\[1, 3, 5\]](#)

An expert is a layer path, not a whole separate assistant

MoE adds parameter capacity by selecting a few feed-forward paths per token. The unselected experts remain part of the stored model.

At an MoE layer, a router computes scores from a token's hidden state. Selection and gating determine which routed experts process it. A shared expert can process the token alongside those routed experts. The combined result continues through the same decoder stack. Attention, embeddings, normalization and output projection still matter.

A generic teaching equation combines expert outputs using weights, but real releases implement gating differently. The saved Llama 4 implementation applies selected sigmoid scores to expert inputs before its nonlinear FFNs, so it should not be described as a literal normalized weighted-output average. DeepSeek has its own scoring, group restrictions and balancing design.

The cited Llama 4 releases select one routed expert plus a shared expert: Scout has 16 routed experts and 109B total / 17B active parameters; Maverick has 128 routed experts and 400B total / 17B active. The original DeepSeek V3 configuration has 256 routed experts, 8 activated routed experts and 1 shared expert. Its first three FFN layers are dense, followed by expert layers.

Active counts estimate participating parameters under the release's convention. They do not turn a 671B stored model into a 37B checkpoint. Serving still places or offloads the full expert set; dispatch, load balance and cross-device traffic affect speed. Experts also need not correspond to recognizable business topics.

Illustrative output-gated MoE: $y = E_{\text{shared}}(x) + \sum_i E_{\text{selected}} g_i E_i(x)$
This equation is not the literal gating rule for every MoE implementation.

Worked example. Generic toy outputs $E_1 = [1, 0]$, $E_2 = [0, 2]$, shared = $[0.2, 0.2]$, and gates $[0.6, 0.4]$ give $y = [0.8, 1.0]$. This illustrates combination only; it is not a simulation of Llama 4's actual router.

Release evidence [\[2, 3\]](#)

Llama 3.1 and Llama 4: what changes inside

The shared next-token idea stays. The cited generations change their feed-forward design, inputs and operating footprint.

Llama 3.1's 8B, 70B and 405B releases are text models with dense decoder computation and GQA. Their advertised context is 128K. A token passes through repeated attention and FFN blocks, then becomes vocabulary logits; its generated continuation is processed in the same autoregressive loop.

Llama 4 Scout and Maverick add image input with early fusion and replace selected feed-forward computation with MoE paths. The decoder can combine visual representations and text while producing text or code outputs. Image understanding is an input capability, not image generation. Their cards advertise 10M context for Scout and 1M for Maverick.

Scout's 109B total / 17B active counts and Maverick's 400B total / 17B active counts describe different expert capacity despite the same published active count. In the saved implementation, each routed token selects one expert while the shared FFN path also runs. Do not equate active parameters with total memory or guarantee equal speed between these checkpoints.

For a business workflow, text-only extraction and document/image analysis have different input needs. The useful comparison is the smallest release and deployment configuration that meet quality, response-time and memory requirements on the actual task. The published context maxima should be load-tested instead of treated as a ready-made production budget.

Dense path: token representation → attention → regular FFN → repeated blocks

Expert path: token representation → attention → selected expert path + shared path → repeated blocks

Worked example. A text-only report summarizer may not need a vision path. A chart-reading assistant does. Both require their own quality and capacity evaluation, even within the same model family.

Release evidence [\[1, 2\]](#)

Gemma 3: local and global attention in one dense decoder

Gemma 3 uses dense feed-forward layers while restricting the attention span in many layers. This is a different efficiency strategy from routing experts.

The inspected Gemma 3 configurations repeat five local sliding-window attention layers followed by a global layer. Local attention sees a bounded nearby span; a global layer can connect the full permitted earlier context. Residual paths and feed-forward transformations still accompany the layers. Repetition through the stack gives representations routes for information to travel across different spans.

The configurations define 1B / 4B / 12B / 27B variants with 26 / 34 / 48 / 62 decoder layers. Query and K/V-head counts are respectively 4/1, 8/4, 16/8 and 32/16. Local windows are 512 tokens for 1B and 1,024 for the other cited sizes. Fewer K/V heads reduce conventional cache state, and the configuration includes Q/K and additional layer normalization.

The 1B configuration has no vision encoder and explicitly supports 32,768 positions. The larger variants add a SigLIP vision path. Pan-and-scan preprocessing can supply extra crops for detail in high-resolution or wide images; it can also add processing work. The inspected 12B and 27B configs explicitly set 131,072 positions. Exact 4B context should be checked against the release's authoritative card and chosen runtime, because this saved implementation's 4B function omits that override.

A local window is not the whole model's maximum context. Global layers still matter for memory, and an advertised limit does not promise perfect recall. This brief makes no independently verified claim about Gemma 3's exact pretraining-token total or corpus mixture.

Attention schedule: local → local → local → local → local → global → repeat
Conventional per-layer cache scales with K/V heads and retained positions.

Worked example. For a contract clause several thousand tokens earlier, a local layer cannot directly read that clause outside its window; a global layer can access it, and intermediate representations can carry information across the stack.

Release evidence [\[5, 6\]](#)

DeepSeek V3 and R1: backbone, latent cache and reasoning training

Original full R1 uses the V3-Base backbone. Its reasoning behavior comes from post-training; it does not represent a wholly different decoder architecture.

The original V3 configuration defines 61 decoder layers and a hidden width of 7,168. The first three FFN layers are dense; later layers use DeepSeekMoE with 256 routed experts, 8 activated experts and one shared expert. V3 and full R1 are published as 671B total / 37B active models.

MLA compresses key/value information through a learned lower-dimensional representation rather than retaining conventional independent K/V tensors for every attention head. The configuration specifies a K/V latent rank of 512 and a separate rotary query/key dimension of 64. Efficient inference uses the latent structure; exact cache allocation and speed depend on the runtime. The rank numbers alone are not a measured end-to-end memory result.

V3 also used multi-token prediction during training. That auxiliary objective does not mean every ordinary deployment necessarily emits several final tokens in one step; serving support and decoding methods must be checked separately.

R1-Zero applied RL directly to V3-Base without preliminary SFT. R1 added cold-start data and an alternating SFT/RL pipeline to improve reasoning and usability. Smaller R1-Distill releases train dense Qwen or Llama students on curated examples; they retain their student architectures and do not become miniature 671B expert backbones. Longer generated reasoning can help some tasks, while increasing tokens and waiting time; it remains output to verify.

Conceptual MLA path: hidden state → compressed K/V representation → attention-compatible transformations

Reasoning adaptation: V3-Base → post-training → R1 behavior

Worked example. Choose a full R1 checkpoint only if your deployment can serve its total footprint. A dense R1-Distill checkpoint is a separate model choice and needs its own task evaluation, not an assumption of identical quality.

Release evidence [\[3, 4\]](#)

Worked storage math: weights are the first line of the budget

A parameter count and a bit width give a useful lower-order storage estimate, but not a complete hardware specification.

For P parameters stored with b bits each, raw weight storage is $P \cdot b/8$ bytes. Use total parameters for an MoE checkpoint, not just its active count. A billion parameters at 16 bits is 2 decimal GB; at 8 bits it is 1 GB; at 4 bits it is 0.5 GB before metadata.

An 8B model therefore has about 16 GB of bare 16-bit weights, 8 GB at 8 bits, or 4 GB at 4 bits. In binary units, 16 billion bytes is about 14.9 GiB. A 671B model at 16 bits would be approximately 1,342 GB of bare weights, not 74 GB computed from its 37B active count. This is a hypothetical uniform representation, not the released V3 file format.

Real quantized checkpoints add scales, packing and possibly higher-precision tensors. Serving adds cache, activations, workspaces and runtime overhead; distribution across devices adds placement constraints. Training adds gradients, saved activations and optimizer state, whose precision and sharding are setup-specific.

For capacity planning, inspect the actual checkpoint files and measure device use with your input lengths and simultaneous requests. Lower precision can reduce memory but does not guarantee faster kernels or unchanged quality. An estimate that barely equals a GPU's nominal memory leaves no useful working-state budget.

Raw weight bytes $\approx P \cdot b/8$

Decimal GB = bytes / 10^9

Binary GiB = bytes / 2^{30}

Worked example. $8B \times 16/8 = 16 \text{ GB} \approx 14.9 \text{ GiB}$. Original full V3/R1: $671B \times 16/8 = 1,342 \text{ GB}$ at hypothetical uniform 16-bit storage, before working state. V3's repository provides FP8 weights, so this formula is a planning comparison, not its download size.

Release evidence [\[1, 3, 4\]](#)

Data, stages and what the published counts mean

A training-token count describes the volume processed at a stage. It is not a count of unique facts, unique documents, or verified permissions.

Pretraining draws on broad sequences of text, code and sometimes image-text material. Filtering, deduplication, language balancing and separating evaluation content change what reaches training. Instruction examples then teach responses; preference and reward signals adjust behavior. The final model is shaped by these stages and objectives together, not by one public dataset name.

The Llama 3.1 card reports 15T+ pretraining tokens from publicly available online data and more than 25 million synthetic fine-tuning examples. Llama 4's card reports about 40T tokens for Scout and 22T for Maverick from a mix of public, licensed and Meta-product data. Original DeepSeek V3 reports 14.8T pretraining tokens. These are publisher disclosures, not an independently reproduced corpus audit.

R1's full models use V3-Base; its reasoning adaptation and student distillation are later training stages. The R1 repository explicitly describes the Qwen-based distillations as fine-tuned with 800K curated samples. An example count at that stage is not a pretraining-token total. One example can itself contain many tokens.

For Gemma 3, this brief uses verified configuration and implementation facts while leaving exact corpus mixture and token totals unasserted. None of the inspected materials supplies a complete reconstructable pretraining manifest for the cited releases. Public corpora such as C4, FineWeb and The Pile can illustrate dataset curation, but they must not be labeled the exact training sources without evidence.

For an internal project, document dataset rights and provenance, remove duplicated and evaluation-overlapping material, version the task examples and retain a holdout. A larger token count alone does not establish better quality for your business task.

Training tokens $\neq$ unique documents $\neq$ post-training examples
Stage, unit and disclosure source must accompany each count.

Worked example. A set of 800,000 prompt-response examples might contain hundreds of tokens per example. Comparing that example count directly with 14.8 trillion pretraining tokens mixes units and training stages.

Release evidence [\[1, 2, 3, 4, 5, 6\]](#)

Worked K/V-cache math and why concurrency matters

The weights can stay the same while context memory grows with longer sequences and more simultaneous requests.

For conventional full-context attention with equal key/value head dimensions, a simple cache estimate is $2 \cdot B \cdot T \cdot L \cdot H_{kv} \cdot d \cdot s$ bytes. The factor 2 counts keys and values; B is simultaneous sequences, T retained tokens per sequence, L layers, H_{kv} K/V heads, d head dimension, and s bytes per cache value. Prompt and generated tokens both contribute to T.

In a hypothetical design with $B = 1$, $T = 8,192$, $L = 32$, $H_{kv} = 8$, $d = 128$ and $s = 2$, the cache is 1,073,741,824 bytes, exactly 1 GiB. At eight equal-length simultaneous sequences it becomes 8 GiB. With 32 K/V heads instead of 8, the one-sequence cache would be 4 GiB under otherwise identical assumptions.

Requests rarely have identical lengths. Replace $B \cdot T$ with the sum of retained lengths when estimating mixed traffic. Account for allocation blocks, extra runtime state and any prefix sharing. Local layers can retain fewer positions in implementations that evict old cache state; global layers still retain longer spans. MLA uses a different cache representation, so this conventional formula should not be applied blindly to DeepSeek.

Concurrency is therefore a workload condition, not a fixed model property. Capacity tests should combine realistic context and output lengths with overlapping requests, including long outliers and queueing.

Conventional K/V bytes $\approx 2 \cdot L \cdot H_{kv} \cdot d \cdot s \cdot \sum_r T_r$
Equal-length special case: $2 \cdot B \cdot T \cdot L \cdot H_{kv} \cdot d \cdot s$

Worked example. Toy configuration: $2 \times 1 \times 8,192 \times 32 \times 8 \times 128 \times 2 = 1$ GiB. Eight such sequences need about 8 GiB of this cache alone; weights and other memory are additional.

Worked serving metrics: fast first output versus total capacity

Prompt length, answer length and overlapping requests affect different parts of the user experience. A single “tokens per second” number cannot describe all of them.

Prefill processes the supplied prompt and prepares cache state. Decode extends the output using that cache. Longer prompts usually increase work before the first token; longer responses add decoding steps. Request queueing and application work can dominate either stage at high load.

Consider an illustrative request with TTFT = 0.80 seconds and 200 generated tokens. If the subsequent 199 tokens average 25 ms each, total model-stream time is about $0.80 + 199 \cdot 0.025 = 5.775$ seconds, before any additional post-processing. After startup, that request’s token rate is about 40 tokens per second.

Aggregate throughput is different. If eight simultaneous requests each sustain 30 output tokens per second during a measured interval, the aggregate is about 240 output tokens per second. This does not establish how long a new request will queue or what its TTFT will be. Batching can improve total work completed while changing per-request latency.

Record input/output token lengths, concurrency, throughput, TTFT, total latency, tail percentiles and failures together. Use the same model version, precision, runtime, hardware and sampling settings when comparing runs. A batch document workflow and an interactive chat interface can reasonably choose different tradeoffs.

Total stream time $\approx$ TTFT + (Noutput – 1) · TPOT

Aggregate output throughput = total emitted output tokens / measured elapsed time

Worked example. Teaching workload: TTFT 0.80 s, 200 output tokens, subsequent TPOT 0.025 s → about 5.775 s. These are invented values, not a benchmark for any model in this brief.

SFT, RLHF and DPO use different forms of feedback

Instruction examples teach a desired response directly. Preference methods train from judgments about alternatives. Neither method turns subjective feedback into a guarantee of truth.

SFT commonly trains the model to give high probability to a supplied target response. A reviewed example can teach formatting, task steps or response style. If the examples are wrong or narrow, the model can learn those weaknesses too.

Classic RLHF collects preferences between answers, trains a reward model to predict those preferences, then applies RL to improve the language model's expected reward. A reference-policy penalty can limit excessive changes. The reward model is an approximation of the collected judgments and can be exploited or misgeneralize.

DPO uses preferred/rejected response pairs directly. A common loss compares the policy's log-probability ratio for the two answers against a fixed reference model, scaled by β . It increases the relative likelihood of preferred answers without the standard separate reward-model training and online RL rollout loop. Exact implementation and data choices still matter.

Preference is a useful behavioral signal for clarity, tone and instruction following. It does not establish that a cited source exists, a SQL query is correct, or a numerical answer follows from the evidence. For consequential tasks, combine preference evaluation with explicit correctness checks and held-out examples. The Llama 3.1 card reports SFT and RLHF; this module's DPO equation explains a general method, not an assertion that every cited release used it.

DPO: $-\ln \sigma(\beta[(\ln \pi_{\theta}(y_{+}|x) - \ln \pi_{\text{ref}}(y_{+}|x)) - (\ln \pi_{\theta}(y_{-}|x) - \ln \pi_{\text{ref}}(y_{-}|x))])$
 y_{+} = preferred response; y_{-} = rejected response; σ = sigmoid

Worked example. Two draft summaries can be ranked for clarity. A separate check should verify their totals and citations; a clearer wrong answer should not be accepted merely because it was preferred.

Release evidence [\[1\]](#)

GRPO, verified rewards and the limits of “reasoning”

Reward-driven training can discover useful solution behavior, but it optimizes the checks and judgments provided, not an unlimited concept of correctness.

A common GRPO setup generates a group of responses to one prompt and scores them. Subtracting the group’s mean reward and dividing by its standard deviation produces relative advantages. A clipped policy objective and a reference-policy constraint then guide updates. Unlike classic actor-critic training, the common formulation estimates this advantage without training a separate value critic.

For rewards $[1, 0, 0, 0]$, the mean is 0.25 and the population standard deviation is approximately 0.4330. The normalized advantages are $[1.732, -0.577, -0.577, -0.577]$. The successful response receives positive relative signal. If every response gets the same reward, mean subtraction gives zero signal; implementations use a small ϵ for numerical stability, and details vary.

A code test can verify its tested properties, and an exact-answer checker can verify a defined answer format. Neither covers every hidden requirement, real-world edge case or explanation step. Training can exploit incomplete checks. Use robust verifiers and independently assess whether gains transfer to new tasks.

The cited R1 repository documents direct RL for R1-Zero and a cold-start plus SFT/RL pipeline for R1. Generated reflection or long reasoning text may correlate with improved performance on some tasks, but it can also contain errors and consume substantial latency. This generic GRPO calculation explains group-relative feedback; the source list is not being used to assert every detail of an uncited training implementation.

$$A_i = (r_i - \text{mean}(\text{rgroup})) / (\text{std}(\text{rgroup}) + \epsilon)$$

This is a simplified group-relative advantage, not the full policy-update objective.

Worked example. Toy rewards $[1, 0, 0, 0] \rightarrow$ advantages approximately $[1.732, -0.577, -0.577, -0.577]$. Winning a narrow checker is evidence for that checker’s task, not universal reasoning ability.

Release evidence [4]

Prompting, RAG, LoRA and distillation change different things

Choose the intervention that matches the problem: available facts, response behavior, model footprint or deployment memory.

Prompting supplies instructions and examples at inference. RAG adds retrieved evidence, useful for current or private facts. Neither ordinarily changes model weights. Their success depends on retrieval quality, context handling and whether the response actually follows the provided evidence.

Full fine-tuning changes many base parameters. LoRA represents a matrix update with two smaller matrices, often while the base weights remain frozen. For a $4,096 \times 4,096$ matrix, a full update has 16,777,216 entries. A rank-8 LoRA update has $8 \cdot (4,096 + 4,096) = 65,536$ trainable entries, about 0.39% of that single matrix's full count. This is not the adapter fraction of an entire model, and the base model still occupies memory.

Distillation trains a student on teacher probabilities or examples, potentially moving useful behavior to a smaller model. Quantization instead changes numeric representations of weights or working state. These methods can be combined, but each needs quality evaluation.

The R1-Distill releases illustrate architecture-preserving students: their dense Qwen/Llama backbones stay dense while learning from curated R1 examples. Distillation does not copy the teacher's expert router, total capacity or every ability. For a workflow, a small student plus accurate retrieval can be a useful candidate, rather than an assumed replacement for the teacher on all tasks.

LoRA: $W_{\text{adapted}} = W_{\text{base}} + (\alpha/r)BA$

Trainable entries for one $m \times n$ matrix: $r(m+n)$, compared with mn for a full update

Worked example. Need this quarter's policy? Retrieve it. Need stable output fields? Improve examples or adapt behavior. Need lower memory? Evaluate quantization or a smaller student. Test the combined workflow on held-out cases.

Release evidence [\[4\]](#)

Evaluate what the workflow needs, including generalization

A convincing model score is useful evidence only when the measured task and operating conditions support your decision.

Define acceptance before testing: correct fields, grounded facts, successful executable output, appropriate uncertainty and tolerable failure severity. Compare released checkpoints on the same versioned task set, with consistent prompts, tools, precision and scoring. Keep development cases separate from a final holdout; repeated tuning against a holdout makes it less independent.

Track correctness and latency together. Long reasoning, more sampled attempts, retrieval and human review can improve acceptance while raising cost. Report the whole workflow, not only the model's best answer. Separate a verified result from a fluent but unchecked explanation.

Generalization means performing on new cases from the intended workflow. Lower loss on training examples does not ensure this. Contamination, duplicated documents, highly similar templates and evaluating on familiar examples can inflate apparent quality. Changes in language, document layout, business rules and data sources can reduce performance after launch.

Use cost per accepted outcome to connect technical quality with business impact. An illustrative \$200 workflow that produces 80 accepted reports costs \$2.50 per accepted report; a \$150 workflow with only 40 accepted reports costs \$3.75. Include retries, human review and idle capacity in the cost boundary. The better option depends on meeting minimum quality and service requirements, not just minimizing a token price.

Acceptance rate = accepted outcomes / attempted tasks

Cost per accepted task = total workflow cost / accepted outcomes

Worked example. Candidate A: \$200 / 80 accepted reports = \$2.50 each. Candidate B: \$150 / 40 accepted reports = \$3.75 each. These are hypothetical workflow totals; define comparable cost and acceptance boundaries.

Inspect the claim. Pin the release.

Architecture and licensing change between releases. This guide identifies its reference checkpoints and distinguishes verified implementation details from illustrative explanations.

01

Meta / Llama 3.1

Model card: dense architecture, sizes, context and data disclosures. Community license is checkpoint-specific.

[Model card](#) · [License](#)

02

Meta / Llama 4

Scout / Maverick card and reference implementation: early fusion, sizes, data and one routed plus parallel shared expert.

[Model card](#) · [Code & license](#)

03

DeepSeek / original V3

Repository and 671B configuration: MLA, three dense / 58 MoE layers, 8 of 256 routed plus one shared, 14.8T tokens and original model license.

[Repository](#) · [Configuration](#)

04

DeepSeek / original R1

R1 / R1-Zero post-training, V3-Base backbone, distillation, generated samples and upstream licensing obligations.

[Repository & license](#)

05

Google / Gemma implementation

Configurations and code verify 1B / 4B / 12B / 27B variants, GQA, local/global cadence, windows and vision path. Whole-request context support must be checked against the specific checkpoint.

[Configurations](#) · [Model code](#)

06

Gemma 3 / documentation

Transformers documentation describes image pan and scan and post-training. Google's report, card and weight terms are further reading; precise corpus statistics are not asserted here.

[Implementation docs](#) · [Report](#) · [Card](#) · [Terms](#)

KAF World Digital · revision 02 · 4 October 2026. Brand references: v10 guide, website consulting systems and portal reading patterns.