

Review

Large Language Models: From Internal Architecture and Distributed Training Optimisation to Adaptation Strategies

Martin Lukáč , František Duchoň , Jakub Ivan , Martin Dekan , Eduard Zelenay, Maroš Kocúr, Izabela Trepáčová, Tomáš Jurov and Radoslav Pšenka 

Institute of Robotics and Cybernetics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, Ilkovičova 3, 841 04 Bratislava, Slovakia

* Correspondence: martin.lukac@stuba.sk

Abstract

This paper presents a unified technical survey of Large Language Models (LLMs), connecting three layers of the modelling pipeline that existing surveys address in isolation: internal architecture, distributed training optimisation, and downstream adaptation. Its organising principle is the dependency between these layers—how a choice at one constrains what remains feasible at the next. The survey examines fundamental mechanisms (tokenisation, scaled dot-product attention, activation functions, and normalisation, including RMSNorm and pre- versus post-normalisation placement), then the engineering of training at scale: data, tensor, and pipeline parallelism, hybrid schemes, mixed-precision training with BF16 and FP8, ZeRO-Offload memory management, activation checkpointing, and compute-optimal scaling laws together with the conditions under which they fail. The adaptation section covers supervised and instruction fine-tuning, a comparison of parameter-efficient methods (LoRA, QLoRA, adapters, prefix and prompt tuning), Reinforcement Learning from Human Feedback with its reward-hacking failure mode, alternatives including DPO, KTO and Constitutional AI, Retrieval-Augmented Generation beyond the basic pipeline, and decoding strategies. Practical configuration guidance is given for 7B, 70B and trillion-parameter regimes. Dedicated treatments of Mixture-of-Experts architectures, long-context modelling, and hardware-aware co-design close the survey, with open challenges classified by origin and severity.

Keywords: large language models; distributed training; parallelism; fine-tuning; retrieval-augmented generation



Academic Editors: Aristeidis Karras and Ioannis Karydis

Received: 30 June 2026

Revised: 2 August 2026

Accepted: 4 August 2026

Published: 6 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Large Language Models (LLMs) have advanced substantially since the introduction of the Transformer architecture [1], progressing from sequence-to-sequence translation systems to billion-parameter generative models trained on multitrillion-token corpora. Surveys such as Minaee et al. [2] and Raiaan et al. [3] catalogue architectural variants and applications; Liu et al. [4] address training and inference with greater systems focus. These works treat tokenisation, distributed training engineering, and downstream adaptation as largely separate concerns; however, leaving engineers who need to understand the full pipeline without a unified reference.

This survey addresses a specific gap: the engineering dependencies across the three layers are rarely traced in a single source. Vocabulary design determines sequence length; it constrains per-device memory allocation and memory constraints determine which

parallelism configuration is feasible and the scale of the resulting training regime determines which adaptation strategy is tractable at deployment. Understanding these dependencies requires that the layers be examined together rather than in isolation.

This paper connects them within a single structured treatment, covering internal mechanisms, distributed training at scale, and adaptation paradigms. Each section is written to expose not only what a given technique does but why it is necessary and how it interacts with other components in the pipeline.

Table 1 positions this work relative to the closest existing surveys. A coverage matrix, however, records what each work addresses, not what it is organised around—and the organising axis determines what a survey can state at all.

Table 1. Comparison of coverage areas in representative Large Language Model (LLM) surveys. Tokenis. = tokenisation, Mem. Opt. = memory optimisation, Scal. Laws = scaling laws, RAG = Retrieval-Augmented Generation. ✓ = primary focus, ○ = partial coverage, – = not covered.

Survey	Tokenis.	Attention	Parallelism	Mem. Opt.	Scal. Laws	Decoding	Fine-Tuning	RAG
Minaee et al. [2]	○	✓	○	–	○	–	✓	○
Raiaan et al. [3]	✓	✓	–	–	–	–	✓	○
Liu et al. [4]	✓	✓	○	○	✓	✓	✓	○
This work	✓	✓	✓	✓	✓	○	✓	✓

Minaee et al. [2] organise by model family and capability. The engineering constraints that determine whether an architecture is trainable at a given scale do not attach to any model family, so a family-based taxonomy has no place to put them. Raiaan et al. [3] organise by taxonomy and application domain, which treats the trained model as the starting point, leaving distributed training as background. Liu et al. [4] are the closest prior work and most systems-aware, but their axis is the model lifecycle—training, then inference—so parallelism and memory optimisation appear as stages the model passes through rather than as constraints that propagate; a lifecycle ordering places a vocabulary decision and a memory ceiling in different chapters with no mechanism for relating them.

None of these works is deficient on its own terms; each competently treats the axis it selects. What none supplies is the *dependency chain*: not what exists at each layer, but what each layer’s choices forbid at the next. Section 2.1.3 shows a vocabulary decision fixing an attention cost, Section 3.6, a bandwidth asymmetry fixing a parallelism degree, Section Deviations from the Compute-Optimal Frontier, an architectural choice invalidating a token budget, Section 7.2.1, a routing decision becoming a communication problem. Table 1 should be read as evidence of that gap rather than as a claim to greater breadth.

Scope and delimitation. This survey is organised around one relation: the dependency by which a choice at one layer constrains what is feasible at the next. Subjects are treated in proportion to their participation in that chain—tokenisation, attention cost, normalisation, parallelism, precision, scaling, and adaptation. Deliberately out of scope, each having dedicated surveys of its own: data curation and its legal status, evaluation methodology, multimodal architectures, agentic systems, production serving infrastructure—request scheduling, autoscaling, multi-tenancy and the operational concerns of running a cluster—beyond the decoding strategies of Section 4 and the inference-efficiency techniques of Section 7.2.3; and interpretability. We claim not broader coverage than existing surveys, but a relation between layers that a broader treatment structurally cannot state.

The main contributions of this paper are as follows:

- A unified, end-to-end technical overview that coherently connects LLM internal architecture, distributed training optimisation, and downstream adaptation strategies within a single reference, addressing the fragmentation identified in the existing literature.
- A detailed treatment of distributed training parallelism strategies (data, tensor, and pipeline) and their hybrid combination, developed through general formulations rather than fixed-size walkthroughs, and analysed in terms of the memory saving each achieves against the communication volume it incurs.
- An analysis of practical optimisation techniques—mixed-precision training with 16-bit brain floating point (BF16) and 8-bit floating point (FP8) trends, ZeRO-Offload, and activation checkpointing—contextualised within neural scaling laws and compute-optimal training principles.
- A comparative examination of LLM adaptation paradigms—fine-tuning, parameter-efficient methods compared side by side, alignment by Reinforcement Learning from Human Feedback together with its alternatives and failure modes, and Retrieval-Augmented Generation (RAG)—identifying the conditions under which each strategy is preferable for downstream deployment.
- Practical configuration guidance derived from the constraints established in the survey: a parallelism selection procedure, a mixed-precision and memory checklist, and the qualitatively different regimes that apply at 7B, 70B and trillion-parameter scale.
- An account of the dependency between these layers—the relation by which a choice at one layer fixes the budget available at the next—which the survey treats as its organising contribution rather than as a by-product of covering each layer in turn.

2. Internal Mechanisms of Large Language Models

2.1. Tokenisation

Tokenisation is a crucial preprocessing step in LLM training that divides text into indivisible units known as tokens. Depending on the specific tokenisation method, these tokens may represent characters, subwords, symbols, or entire words. Modern LLMs predominantly use subword tokenisation strategies, two widely adopted instances are Byte Pair Encoding (BPE) and the SentencePiece unigram model [5,6].

2.1.1. Byte-Pair Encoding (BPE)

Byte-Pair Encoding (BPE) is a compression-inspired subword segmentation algorithm widely adopted for neural machine translation. The vocabulary is initialised with individual characters; the algorithm then repeatedly identifies the most frequent adjacent symbol pair in the corpus and replaces all its occurrences with a merged symbol, until the target vocabulary size is reached. At inference, the learned merge rules are applied sequentially to segment any input [7,8].

Its advantage is balancing vocabulary size against tokens per sentence. Because merges follow corpus frequency, high-frequency substrings consolidate early, so common words become single tokens whereas infrequent or morphologically complex ones split into several units—keeping token counts manageable even with a compact, fixed-size vocabulary, which matters for decoding efficiency [7,8].

A notable drawback of BPE as originally formulated is that its merge procedure is fully deterministic: each input string maps to exactly one segmentation with no associated probability score. Kudo [5] identified this as an obstacle to subword regularisation, which requires a probability distribution over alternative segmentations $P(x | X)$ in order to inject segmentation diversity as a form of data augmentation during training, and proposed the unigram language model as an alternative that supplies such a distribution by construction.

This limitation, however, proved to be one of the standard *applications* of BPE rather than of the algorithm itself. Provilkov et al. [9] observe that a BPE merge table already implies many possible segmentations of a word, and introduce BPE-dropout, which stochastically skips merges during training so that one merge table yields varied segmentations. Standard deterministic BPE is retained at inference, leaving the deployed tokeniser unchanged; the method reports gains of up to 3 BLEU over BPE and up to 0.9 BLEU over previous subword regularisation, attributed to improved compositionality and robustness to segmentation errors rather than to vocabulary coverage [9]. The choice between BPE and a unigram model, therefore, need not be decided by regularisation capability, as the earlier literature implied: both admit stochastic segmentation.

Applied to *mimimississippi*, three merges ($mi \rightarrow A$, $ss \rightarrow B$, $Bi \rightarrow C$) compress the string from 15 symbols to 8. Two properties follow. Each step replaces *all* non-overlapping occurrences of the selected pair simultaneously, so merge order is fixed by corpus frequency rather than position, and merges compose— C expands to ssi only by way of B —so the list must be replayed in its original order for the segmentation to be reproducible. This ordering requirement is what makes the learned merge table, not merely the vocabulary, an inseparable part of a released tokeniser.

2.1.2. Subword Tokenisation

Subword tokenisation divides words into smaller meaningful units, addressing words absent from the vocabulary (out-of-vocabulary, OOV). Because languages build words from morphemes, an unfamiliar word may still decompose into familiar components: under the WordPiece strategy [10,11], *unhappiness* segments as *un* + *happi* + *ness*, each likely present in the vocabulary, allowing the model to infer meaning without prior exposure to the whole form and to reuse those units for related forms such as *unhappily* [6]. This mitigates OOV while improving generalisation across word variations [7,10].

Limitations of subword tokenisation. Despite its benefits, subword tokenisation is not without drawbacks. The segmentation of a word depends on the training corpus; infrequent words in the pre-training data may still be split into unintuitive or semantically opaque subword units. Furthermore, subword boundaries do not always align with morphological structure, which can have an effect on performance in morphologically rich languages [5,6]. Finally, because the vocabulary is fixed at training time, extending the model to new domains or languages typically requires either retraining the tokeniser or accepting a higher rate of rare-token decomposition [5,6,10].

2.1.3. Tokeniser Design as an Engineering Decision

Tokenisation is usually presented as preprocessing. In a production pipeline, the tokeniser fixes the units in which every downstream cost is denominated, with two quantifiable consequences.

Vocabulary size and sequence compression. Production vocabularies have grown well beyond those of early BPE systems for machine translation. The Llama 3 tokeniser uses 128k tokens—100k inherited from *tiktoken* plus 28k added to support non-English languages—raising compression from 3.17 to 3.94 characters per token on English data relative to Llama 2 [12]. The significance is sequence length, not storage: fewer tokens give a shorter n , and by the $O(n^2 \cdot d)$ cost of Section 2.2 a $\approx 24\%$ gain in characters per token cuts the attention term for a fixed document by substantially more than 24%. Vocabulary size thus trades a linear cost—larger embedding and output projections, replicated or sharded across devices—against a quadratic saving. A vocabulary decision taken before training constrains the context length a given memory budget can support.

Multilingual disparity. The same mechanism produces asymmetry across languages. Petrov et al. [13] show the same text translated into different languages yields tokenised lengths differing by up to a factor of 15, with consequences for the cost of commercial language services, latency, and the content supplyable as context. The last matters most here: the effective context window is not a property of the architecture but a function of the input language, so a speaker of an under-represented language receives materially less usable context from the same nominal window. Adding non-English vocabulary, as Llama 3 does, mitigates but does not remove this, at the cost of the embedding growth noted above.

SentencePiece [6] addresses the same problem at the level of the tokeniser’s interface rather than its vocabulary. It is a language-independent subword tokeniser and detokeniser that trains directly from raw sentences, which removes the dependence on a language-specific pre-tokeniser that word-based pipelines require and that does not exist for languages written without whitespace [6]. Whitespace is escaped with a meta symbol (U+2581) and treated as an ordinary character, so that decoding is the exact inverse of encoding and the normalised text is recoverable from the token sequence [6]. The package implements both byte-pair encoding and the unigram language model over this interface [6], which is why the choice of segmentation algorithm and the choice of a language-independent input representation are independent decisions, and why a multilingual model may adopt BPE merges while retaining the raw-text handling that makes the tokeniser portable across scripts.

2.2. Attention Mechanisms

Transformers process all tokens in an input sequence simultaneously, without sequential recurrence. Because the attention module does not inherently capture positional information, positional encodings are added to token embeddings to provide the necessary sequence order. These positional embeddings can be absolute, relative, or learned; contemporary large models predominantly adopt rotary position embeddings, whose mechanism and long-context consequences are treated in Section 7.2.2 [14]. Attention assigns weights to input tokens based on their relevance to each other, allowing the model to focus on the most informative parts of the input. In Transformers, attention is computed by making the input the query, the key, and the value representations. **The relevance between any two tokens is quantified as a scaled dot product of their query and key vectors, and these scores are used to form a weighted combination of the value vectors [1].**

Introduction of Attention Scores

Each input token is projected into a query Q , a key K , and a value V : the query represents what the token is looking for, the key what it offers, the value the information it carries. For a sequence of n tokens, let Q_i, K_j, V_j denote the query of token i and the key and value of token j ($1 \leq i, j \leq n$). The attention score is the scaled dot product [1]:

$$\text{score}(Q_i, K_j) = \frac{Q_i K_j^\top}{\sqrt{d_k}},$$

where d_k is the key dimensionality. The division by $\sqrt{d_k}$ is necessary because for large d_k the dot products take large absolute values, saturating the softmax and driving its gradient to vanish [1]. Higher scores indicate stronger relevance. The scores are normalised by softmax and used to weight the value vectors [1,15]:

$$\text{Attention}(Q_i, K, V) = \sum_j \alpha_{ij} V_j, \quad \alpha_{ij} = \text{softmax}_j \left(\frac{Q_i K_j^\top}{\sqrt{d_k}} \right),$$

where α_{ij} represents how much token j contributes to the updated representation of token i . Because the softmax normalisation makes the weights α_{ij} non-negative and sum to one over j , each output is a convex combination of value vectors: query–key alignment alone determines which tokens dominate the updated representation, and the projections producing Q , K , and V are what the model learns.

Two properties of this formulation govern the constraints developed throughout the paper. First, every query attends to every key, giving self-attention a complexity per layer of $O(n^2 \cdot d)$ for sequence length n and dimension d , against $O(n \cdot d^2)$ for a recurrent layer [1]. Self-attention is cheaper only while $n < d$; beyond that crossover the quadratic term dominates, which is why context length rather than parameter count binds attention cost, and why the long-context methods of Section 7 target this term. Second, self-attention requires $O(1)$ sequential operations against $O(n)$ for a recurrent layer, whose inherently sequential structure prevents parallelisation within a training example [1]. The absence of recurrence is the precondition for the distributed strategies of Section 3, which depend on partitioning work within a sequence across devices.

2.3. Activation Functions

Activation functions introduce the nonlinear transformations that allow neural networks to model complex, non-linear mappings. The critical engineering constraint is compatibility with backpropagation: during training, gradients are computed by repeatedly multiplying partial derivatives across every layer via the chain rule. Activation functions that compress outputs into a bounded range—such as sigmoid, which maps all real inputs to $(0, 1)$ —cause these products to shrink exponentially as they propagate toward earlier layers. This phenomenon, known as the *vanishing gradient problem*, causes weight updates in early layers to become negligibly small, effectively stalling learning and making very deep networks untrainable without careful activation function selection [16]. Modern LLMs, therefore, use GELU [17] and Swish activations, which maintain non-zero gradients over a wider input range and do not saturate as sigmoid and tanh do. GELU weights inputs by their value rather than gating them by sign, and is defined as $x\Phi(x)$ for the standard Gaussian cumulative distribution function Φ [17]. Contemporary models more often replace the feed-forward activation entirely with a gated linear unit: Shazeer reports that the GELU and SwiGLU variants give the best pre-training perplexities among the alternatives tested [18], and SwiGLU is the form adopted in LLaMA [19]. Table 2 summarises the most commonly used activation functions and their recommended use cases.

Table 2 records a single constraint applied repeatedly rather than a list of options: every function marked problematic is so for one reason—Step is not differentiable and thus incompatible with backpropagation, whereas Sigmoid and Tanh saturate and attenuate the gradient [16]. For LLMs, the “Use” column has consequently collapsed to a single row. A Transformer stack is deep by construction and its trainability depends on gradients surviving that depth—the same requirement motivating the normalisation placement of Section 2.4—so saturating functions are excluded from hidden layers by mechanism rather than preference, leaving the non-saturating family, of which Swish is identified as suited to very deep networks [16]. Activation choice and normaliser placement are usually presented separately but answer to one gradient-flow constraint.

Table 2. Activation function recommendations in deep neural networks [16].

Function	Comment	Use
Step	Not compatible with backpropagation	Rarely
Sigmoid	Vanishing gradient, not zero-centered	Boolean gates
Tanh	Vanishing gradient	RNNs
Softsign	Rarely used	n/a
ReLU	Popular, may die rarely	Hidden layers
LReLU	Like ReLU, avoids dying	If ReLU dies
Maxout	Advanced, more expensive	Last resort
Softplus	Smooth ReLU, slower	Rarely
Swish	Similar to LReLU, may help deep nets	Very deep networks
GELU	Smooth, weights inputs by value [17]	Transformer feed-forward layers
SwiGLU	Gated linear unit; best perplexity among GLU variants [18]	Modern LLM feed-forward layers
SoftMax	For output layer	Classification
OpenMax	Output layer for open classes	n/a

2.4. Normalisation Techniques in Deep Neural Networks

Training state-of-the-art deep neural networks is often constrained by high computational costs. A primary strategy for mitigating prolonged training times involves normalising the activities of neurons [20,21].

Batch Normalisation (BN) was introduced to address what Ioffe and Szegedy term *internal covariate shift*: the distribution of each layer’s inputs changes during training as the parameters of the preceding layers change, so every layer must continually re-adapt to a moving input distribution [21]. BN counters this by standardising each neuron’s pre-activation using statistics—mean and variance—collected over a mini-batch of training examples, after which a learned scale γ and shift β are applied before the non-linearity, restoring the representational capacity that normalisation alone would remove. The reported benefits are substantial: the method permits much higher learning rates and less careful initialisation, and it acts as a regulariser that in some cases eliminates the need for Dropout [21]. Quantitatively, the authors report reaching the same accuracy as their baseline with 14 times fewer training steps, and an ensemble attaining 4.9% top-5 validation error (4.8% test error) on ImageNet classification, exceeding the accuracy of human raters [21].

These gains are paid for with a dependence on batch composition that carries consequences beyond accuracy. The transform $\text{BN}_{\gamma,\beta}(x)$ depends not only on the training example but on the other examples in the mini-batch [21], so it is not a per-example function and its behaviour degrades when batches are small. It also requires a second computational path at inference: because the output must depend deterministically on the input alone, the trained network substitutes population statistics, accumulated as moving averages during training, for the mini-batch statistics used while learning [21]. Application to recurrent models is correspondingly awkward, and the original work identifies it only as a direction for future study [21]. In distributed training these properties compound, since the statistics are computed over whichever shard of the batch a device holds, coupling examples that data parallelism has deliberately separated—which is one reason the Transformer stacks surveyed here normalise per example rather than per batch.

Layer Normalisation (LN) was introduced to overcome these limitations. Rather than pooling statistics over a batch, LN computes the mean and variance across all neurons in the same layer for each individual training example, applying the same adaptive scale and shift as BN. Because the normalisation statistics depend only on a single sample, the computation is identical at training and inference time. This property also makes LN well-suited to recurrent architectures, where the statistics are recomputed independently at every time step without requiring knowledge of the batch. In practice, LN stabilises

hidden-state dynamics in recurrent networks and reduces training time compared to earlier normalisation schemes [20].

The BN/LN contrast, however, no longer describes the choices made in contemporary Transformer stacks. Two further decisions—what the normaliser computes, and where it sits relative to the residual connection—affect the trainability of deep models more than the BN/LN distinction does.

RMSNorm: discarding re-centering. LN confers two invariances: re-centring (subtracting the mean) and re-scaling. Zhang and Sennrich [22] argue that re-centring can be discarded; their Root Mean Square Layer Normalisation (RMSNorm) normalises by the root mean square alone, thereby retaining re-scaling invariance and implicit learning-rate adaptation while removing one reduction over the feature dimension. They report a running-time reduction of 7% to 64% depending on the model, at performance comparable to LN [22]. The saving is largest where normalisation occupies a large share of layer cost, and it is memory-bandwidth-bound rather than compute-bound—precisely the resource Section 3 identifies as scarce.

Pre-LN versus Post-LN. The original Transformer places normalisation between residual blocks (Post-LN). Xiong et al. [23] analyse both placements at initialisation using mean field theory: for Post-LN the expected gradients of parameters near the output layer are large, which is what necessitates learning-rate warm-up, whereas Pre-LN—normalisation inside the residual blocks—yields gradients well behaved at initialisation, permitting warm-up to be removed and giving comparable results with significantly less training time and hyper-parameter tuning [23].

This matters beyond convergence speed. Warm-up is a hyper-parameter requiring re-tuning whenever depth, width, or batch size changes—exactly what changes when a model is scaled or re-partitioned—so Pre-LN decouples the optimisation schedule from the parallelism configuration. The two choices are taken together in practice: LLaMA normalises the input of each sub-layer rather than its output, explicitly for training stability, using RMSNorm [19].

3. Training Strategies for Large Language Models

Graphics Processing Unit (GPU) memory is the primary bottleneck in large-scale LLM training. A model with 70 billion parameters requires approximately 140 GB of storage in BF16 for weights alone. Adam optimiser states (momentum and variance in 32-bit floating point (FP32), totalling 8 bytes per parameter) add a further 560 GB, raising the total to approximately 700 GB—far exceeding the capacity of any single accelerator [24]. Three complementary strategy families address this constraint: distributing computation and parameters across devices (parallelism), reducing the numerical precision of tensors (mixed-precision training), and migrating memory-heavy artefacts to cheaper storage tiers (offloading and activation checkpointing).

3.1. Data Parallelism

In data parallelism, a parameter server maintains the model parameters [25]. Each worker (GPU) independently reads its own distinct shard of the training data from shared storage. It does not receive data from the parameter server. After synchronising the model parameters with broadcast, each GPU performs a full forward and backward pass on its data, producing local gradients. These gradients are subsequently aggregated—either through the parameter server or through all-reduce collectives—so that all devices apply identical parameter updates in each iteration [4].

Distributed data parallelism does not have the parameter server and instead of that it relies on all-reduce operations on gradient information, ensuring that all GPUs obtain

identical gradient values. The results of the all-reduce are communicated to each device, enabling each GPU to update its optimiser state independently. After each update, the model parameters, gradients, and optimiser history remain consistent across all GPUs [4].

Figure 1 illustrates the scheme. The paradigm is hardware-agnostic: the primitives (all-reduce, broadcast) generalise to TPUs, AMD Instinct GPUs, and Graphcore IPUs, and frameworks such as PyTorch Distributed Data Parallel (<https://pytorch.org>), JAX (<https://jax.dev>), and DeepSpeed (<https://www.deepspeed.ai>) abstract over them via collective libraries (NCCL, MPI, Gloo). Dividing a batch across devices reduces the per-GPU batch size, and with it the intermediate activations, by a factor equal to the device count. But the reduction stops there: parameters, gradients, and optimiser state must be stored in full on every device regardless. Even with a single sample per GPU and no intermediate activations, a device may, therefore, still be unable to hold the model [4]—which is what forces the model-partitioning strategies that follow.

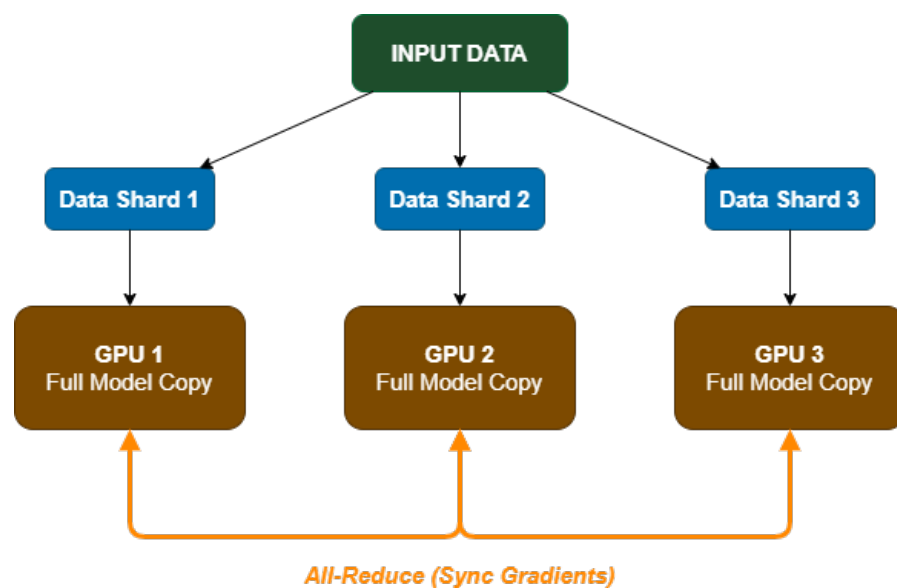


Figure 1. Data parallelism training scheme [4]. Each worker independently reads its own data shard. Forward and backward passes are executed locally on each device, after which gradients are synchronised across all workers via an all-reduce collective operation. The parameter server variant (**top**) routes gradients through a central server; the distributed data parallelism variant (**bottom**) performs all-reduce directly between peers, eliminating the server bottleneck. **Hyperparameters governing utilisation:** the number of workers N sets the gradient all-reduce volume, which is proportional to the parameter count and is incurred *once per optimisation step*—in contrast to tensor parallelism (Figure 2), whose collectives recur at every layer. Per-device batch size B/N determines activation memory; the model state (parameters, gradients, optimiser states) is replicated in full on every device and is, therefore, *independent* of N , which is why data parallelism alone cannot train a model exceeding single-device memory [4].

3.2. Model Parallelism—Tensor Parallelism

A note on terminology is needed before the mechanism, because the two terms are often used loosely. *Model parallelism* is the umbrella category: any scheme that partitions the model itself across devices, as opposed to data parallelism, which replicates the model and partitions the batch. It has two forms, distinguished by the axis along which the partition is taken. *Tensor* (or intra-layer) model parallelism splits the weight matrices *within* a layer, so every device participates in every layer and the partial results must be combined by a collective at each one. *Pipeline* (or inter-layer) model parallelism, treated in Section 3.4, assigns *whole layers* to different devices, so a device is active only when the computation reaches its stage. Tensor parallelism is, therefore, a species of model parallelism rather than

a synonym for it, and the abbreviations used throughout this survey follow that taxonomy: TMP for tensor model parallelism and PMP for pipeline model parallelism. The remainder of this subsection treats the tensor case.

An efficient intra-layer tensor parallelism scheme for Transformer models was formalised in Megatron-LM [26] to **reduce memory usage per device**. In this approach, the structure of model parallelism can be illustrated using a typical **linear layer in a Transformer**. The parameters of the linear layer form a matrix of size $A \times B$, and the input to the layer is a vector of size $B \times 1$. This computation is expressed as $\mathbf{y}_{A \times 1} = \mathbf{W}_{A \times B} \mathbf{x}_{B \times 1}$, where the result $\mathbf{y}$ is a vector of dimension $A \times 1$, not a matrix. The weight matrix can be horizontally partitioned into multiple segments based on the properties of matrix multiplication, with each segment having dimensions $\frac{A}{n} \times B$. Using the distributive property, the input vector x_B is applied to each segment independently, producing partial outputs [4,26].

This partitioning enables the distribution of the linear layer's parameters across several GPUs, as visualised in Figure 2. It is essential, however, that all GPUs receive identical input data; unlike data parallelism, the input batch is not divided but replicated across devices. Each GPU processes its corresponding matrix segment together with the shared input, generating a partial result. These partial outputs must subsequently be concatenated using an all-gather operation and shared across all GPUs to obtain the final output of the linear layer [4].

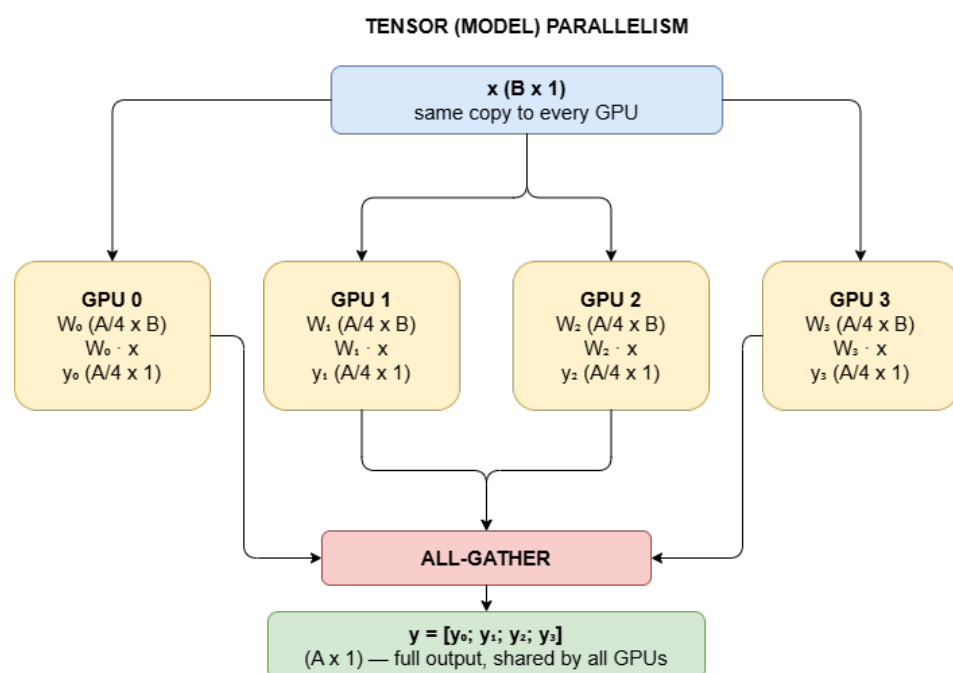


Figure 2. Tensor (model) parallelism scheme [4]. The weight matrix $\mathbf{W}_{A \times B}$ is horizontally partitioned into n equal shards, each of size $\frac{A}{n} \times B$, assigned to separate GPUs. The full input vector $\mathbf{x}_{B \times 1}$ is replicated across all devices (not sharded). Each GPU computes a partial output $\mathbf{y}_i$ of dimension $\frac{A}{n} \times 1$. An all-gather collective then concatenates the partial results to produce the full output $\mathbf{y}_{A \times 1}$, which is shared by all GPUs before the next layer. **Hyperparameters governing utilisation:** the parallel degree n divides per-device weight memory by n , whereas the communication volume needed to reconstruct $\mathbf{y}$ is on the order of the output dimension A and is essentially independent of n . The Megatron-LM formulation incurs two all-reduces in the forward path and two in the backward path per transformer layer [26]. Because these lie on the critical path of every layer, utilisation is bounded by interconnect bandwidth rather than by n : Shoeybi et al. report 300 GB/s intra-server via NVSwitch against 100 GB/s inter-server via InfiniBand [26], which is why n is in practice bounded by node size (Section 3.6).

Generalisation to n -way partitioning. Under n -way tensor parallelism, $\mathbf{W}_{A \times B}$ is partitioned row-wise into n shards of size $\frac{A}{n} \times B$, one per device, so per-device weight memory falls by a factor of n . The full input $\mathbf{x}_{B \times 1}$ is replicated; each device computes a partial $\mathbf{y}_i = \mathbf{W}_i \mathbf{x}$ and an all-gather concatenates the n partials into the $\mathbf{y}_{A \times 1}$ every device needs for the next layer. The pattern applies to every partitioned linear layer (Q, K, V projections, MLP gate/up, output projection). In the backward pass, weight gradients are local, whereas $\partial L / \partial \mathbf{x}$ requires an all-reduce across the group [4,26]. Output-dimension sharding in isolation, therefore, costs one all-gather per layer; Megatron-LM avoids that cost by pairing it with an input-dimension (row-parallel) second GEMM, whose partial sums are combined by a single all-reduce, which is why the collective counts quoted below are all-reduces rather than all-gathers [26].

The resulting trade-off. Memory saving scales as $1/n$, but collectives are required at every partitioned layer: Megatron-LM performs all GEMMs in a transformer layer using two all-reduces forward and two backward [26]. The volume each device must receive to reconstruct $\mathbf{y}$ is on the order of the output dimension A , essentially independent of n . Raising n , therefore, buys a linear memory reduction at approximately constant per-layer communication cost—favourable only while the interconnect hides that cost behind computation.

Placement, not device count, is thus decisive, because bandwidth is strongly asymmetric: Shoeybi et al. report 300 GB/s between GPUs inside a server via NVSwitch against 100 GB/s between servers over eight InfiniBand adapters [26]. Since these collectives lie on every layer's critical path rather than being amortised once per step, a tensor-parallel group spanning a server boundary pays the slower link at each. This is why the tensor-parallel group fits a single node, and why tensor parallelism is combined with other strategies beyond that boundary rather than scaled further—the hybrid schemes of Section 3.6.

3.3. Pipeline Parallelism

Pipeline parallelism is shown in Figure 3 and can be formalised as follows. Consider a Deep Neural Network (DNN) with L sequential layers, where the i -th layer ($1 \leq i \leq L$) holds parameters θ_i , and let f_i and b_i denote its forward and backward functions, respectively. The complete forward pass F and backward pass B are then compositions across all layers [27,28]:

$$F = f_L \circ \dots \circ f_2 \circ f_1$$

$$B = b_1 \circ \dots \circ b_{L-1} \circ b_L$$

Pipeline parallelism partitions the L layers into D contiguous, mutually exclusive groups called stages, one per GPU [27,28]:

$$\{\text{stage}_1, \text{stage}_2, \dots, \text{stage}_D\}, \text{ with } \text{stage}_i \cap \text{stage}_j = \emptyset \text{ for all } i \neq j.$$

Each GPU updates the parameters of its stage. Communication between neighbouring stages carries exactly two kinds of data: activations (layer outputs) forward, and gradients backwards [27–29].

3.4. Performance Limitations of Naive Pipeline Schedule

After a GPU completes its forward step, it must wait until all downstream GPUs have completed both their forward and backward steps before beginning its own backward step [27,28]. This dependency chain creates an asymmetry in waiting time: GPUs at earlier stages idle longest, and at any moment only one GPU computes whereas the rest sit in bubbles—severe under-utilisation [27,28].

Micro-batching subdivides the mini-batch to fill the pipeline more densely, enabling the 1F1B (one-forward-one-backward) schedule and interleaved stage assignments [27–30]. These substantially reduce idle time, but eliminating the bubble at scale without prohibitive communication or memory overhead remains open.

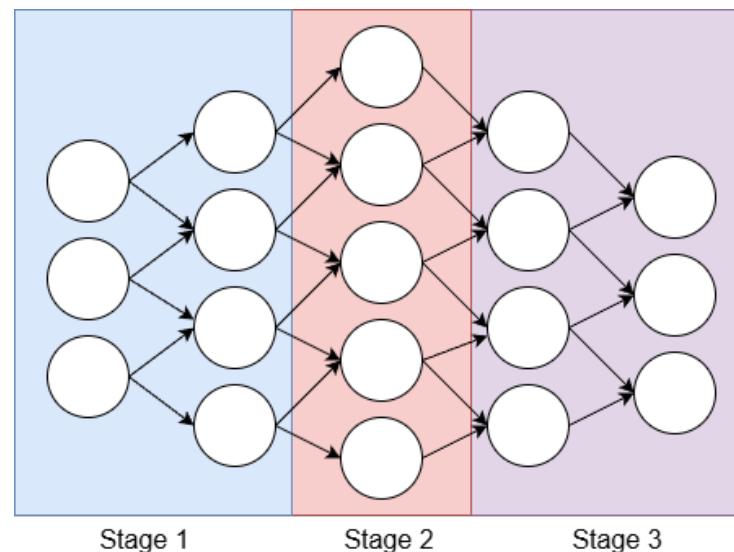


Figure 3. Pipeline parallelism: model partitioning [27]. The DNN layers are split into $D = 3$ non-overlapping stages, each assigned to a dedicated GPU. Arrows indicate the two data transfers required at stage boundaries: layer outputs (forward direction) and gradients (backward direction). **Hyperparameters governing utilisation:** with D stages and a single mini-batch, exactly one of D devices is active at any step, so utilisation is $1/D$ —here $1/3$, and *worsening* as D grows, which is the central obstacle to scaling depth-wise. Splitting the mini-batch into M micro-batches amortises this cost: GPipe gives the resulting bubble overhead as $O\left(\frac{D-1}{M+D-1}\right)$ over M micro-steps, and reports it to be negligible when $M \geq 4 \times D$ [30]. This is the quantitative motivation for micro-batching and the 1F1B schedule discussed in Section 3.4, and it makes M the primary tuning knob for pipeline utilisation [27–30]. Communication at stage boundaries carries only activations and gradients, a volume far below the per-layer collectives of Figure 2, which is why pipelining rather than tensor parallelism is used across the slower inter-node link.

3.5. Mixed Precision Training

Background. *Quantisation* represents parameters, activations, or gradients in fewer bits than FP32. Halving bit-width approximately halves memory footprint and interconnect transfer volume, and on hardware with native support can double arithmetic throughput—at the cost of reduced numerical range and precision. Two strategies are distinguished: *quantisation-aware training*, applying reduced precision during the forward and backward passes with compensating mechanisms, and *post-training quantisation*, compressing a trained model purely for inference. The technique below is quantisation-aware training, applying 16-bit floating point (FP16) selectively while maintaining a full-precision copy where numerical stability requires it [31].

The pre-training of large language models has driven research towards optimisation strategies, particularly focusing on memory management and computational efficiency. Conventionally, the default representation for training parameters is the 32-bit floating-point number (FP32). The use of 16-bit floating-point numbers (FP16 and BF16) has since become an established and widely adopted practice for reducing memory used and communication overhead in large-scale training [31–33].

The primary motivations for adopting FP16 include the substantial reduction in memory used to store model parameters and intermediate tensors, as well as the lower

communication volume between computational units, which lowers communication overhead. Although FP16 has a narrower numerical range and reduced effective precision [31], computations performed in this format are generally faster than those using FP32. Moreover, in typical training scenarios, the magnitude of model parameters remains within the representable bounds of FP16, making the initial conversion both feasible and appropriate [4].

Figure 4 ranks four precision configurations identically across all three metrics: pure bfloat16 performs worst, bfloat16 with random rounding improves somewhat, and the mixed-precision configuration with a maintained float32 optimiser state closely matches full FP32 [31,33]. The consistency of that ranking across an unrelated loss, an accuracy benchmark, and a perplexity measure is what makes the result load-bearing: the FP32 master copy is not a workaround but a quantitatively necessary component for preserving training quality.

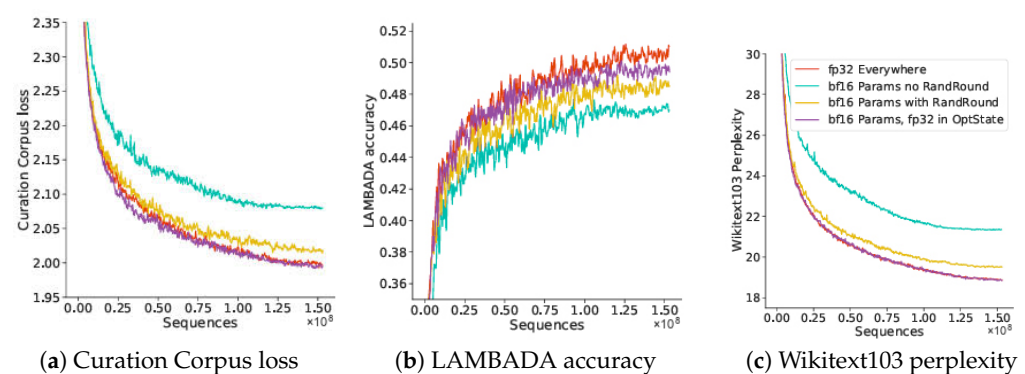


Figure 4. Impact of numerical precision on training quality for a 417M-parameter model, using the mixed-precision methodology of Micikevicius et al. [31] as reported in [33]. Three evaluation metrics are shown: (a) Curation Corpus validation loss [33], (b) LAMBADA next-word prediction accuracy [33], and (c) Wikitext103 perplexity [33]. Each plot compares four configurations: full FP32 (baseline), BF16 without random rounding, BF16 with random rounding, and BF16 with an FP32 master copy maintained in the optimiser state. The results consistently demonstrate that maintaining an FP32 optimiser state copy (mixed-precision) closely matches full-precision quality across all three metrics, whereas pure BF16 without compensating mechanisms creates a measurable accuracy degradation.

Underflow, and the mixed-precision solution. The parameter update $\Delta P \approx \text{gradient} \times \text{learning rate}$ is where FP16 fails, and Micikevicius et al. [31] identify two distinct mechanisms. First, representability: any value whose magnitude is smaller than 2^{-24} becomes zero in FP16, and they report that approximately 5% of weight gradient values have exponents smaller than -24 [31]. Second, and more subtly, alignment: even an update that is representable in FP16 can be lost during the addition itself, because when the magnitude of a normalised weight is at least 2048 times larger than that of its update, the right-shift required to align the exponents drives the update to zero [31]. Underflow is, therefore, not merely a range problem but an accumulation problem, which is why simply rescaling the update would not resolve it.

The remedy follows from that diagnosis. An FP32 master copy of the weights is maintained and updated with the weight gradient during the optimiser step, and in each iteration an FP16 copy of the master weights is used in the forward and backward pass [31]. FP16 arithmetic is thus confined to the compute-dominant passes and FP32 reserved for accumulation—the principle that Section FP8 Training at Scale: A Case Study shows recurring unchanged two hardware generations later.

Current trends: BF16, FP8, and beyond. More recent hardware generations have introduced additional reduced-precision formats. The bfloat16 (BF16) format preserves the 8-bit exponent of FP32, providing the same dynamic range while halving storage. Its adoption is illustrated by BLOOM: after numerical instabilities caused irreversible training divergences under IEEE float16—whose very limited dynamic range can cause overflows—final training was performed in bfloat16 mixed precision, which proved to solve the instability problem [32]. Even more aggressively, an 8-bit floating-point (FP8) interchange format has been proposed with two encodings, recommending E4M3 for weight and activation tensors and E5M2 for gradient tensors, validated on models up to 175B parameters [34]. The trend towards lower-precision numerics is ongoing, with 4-bit formats—FP4 and the integer INT4, alongside the NF4 encoding described below—explored primarily for inference and for parameter-efficient fine-tuning rather than for the training arithmetic itself. The distinction matters for how this section connects to Section 5: 4-bit formats are used there to reduce the *residency* cost of a frozen model, not the precision of its updates, as in QLoRA’s 4-bit NF4 quantisation of the frozen base, whereas the low-rank adapters remain in higher precision [35]. Post-training quantisation for deployment is treated separately in Section 7.2.3.

FP8 Training at Scale: A Case Study

The FP8 literature long consisted of format proposals and small-scale validations, leaving open whether FP8 could carry a frontier-scale pre-training run. DeepSeek-V3 [36] gives the first detailed public account of one, and is instructive for what it declines to place in FP8.

What stays in high precision. Most core computation kernels—the GEMMs—run in FP8, whereas the embedding module, output head, Mixture-of-Experts (MoE) gating modules, normalisation operators, and attention operators are retained in BF16 or FP32, as are master weights, weight gradients, and optimiser states [36]. This generalises the principle of Micikevicius et al. [31] two hardware generations earlier: reduced precision for the compute-dominant, error-tolerant matrix multiplications, full precision wherever values are *accumulated* or a small perturbation is amplified by a subsequent non-linearity. The FP32 master copy of Section 3.5 and DeepSeek-V3’s high-precision optimiser state are the same mechanism answering the same underflow problem, three formats apart.

Fine-grained scaling. Per-tensor scaling proves insufficient at this scale. DeepSeek-V3 scales sub-tensor: activations on a 1×128 tile basis (per token, per 128 channels); weights on a 128×128 block basis [36]. Validated against a BF16 baseline at two model scales over approximately 1 trillion tokens, relative loss error stayed below 0.25% [36]—FP8 pre-training is viable at scale, but only with a scaling scheme far more elaborate than the format proposals imply.

Where FP8 still fails. The obstacles are hardware-level [37]. FP8 uses constrained accumulation precision in Tensor Cores, affecting stability for large models: Hopper maintains only 13 fraction bits for addition after alignment. The fine-grained quantisation that makes FP8 tractable itself incurs significant dequantisation overhead, as Tensor Cores provide no native support for it [37]. The authors’ recommendations to vendors—configurable accumulation precision, native fine-grained quantisation, and dedicated communication co-processors [37]—state precisely why sub-BF16 training is not routine: the arithmetic exists, the accumulation and scaling primitives do not. This is the concrete content of the FP8 instability noted in Section 7.

Stabilising FP8 in practice. The published record supports four concrete precautions, and it is worth separating what is established from what is left to implementations. First, format selection is per tensor role rather than global: Micikevicius et al. recommend E4M3

for weight and activation tensors and E5M2 for gradient tensors [34], the wider exponent being spent where dynamic range matters more than mantissa precision. Second, a scaling factor is not optional and not global: because some networks cannot use one exponent bias for all tensors of a type, a per-tensor adjustment is required, chosen so that the maximum magnitude in the tensor approaches the maximum representable magnitude [34]. Third, outliers must be handled by construction rather than by exception. Values that overflow saturate to the maximum representable value rather than being skipped, and overflow is far likelier than at 16 bits because the dynamic range is narrower [34]; a single outlier, therefore, compresses every other value in its scaling group. This is the motivation for the finer granularity DeepSeek-V3 adopts— 1×128 activation tiles and 128×128 weight blocks [38]—which bounds the number of values any one outlier can distort. Fourth, precision must be preserved wherever values are accumulated or updated rather than merely stored: master weights, gradients and optimiser states are retained above FP8, and the gating of a Mixture-of-Experts layer is kept in high precision because a quantisation error that changes expert selection is a discrete rather than a proportional error [38].

Two caveats limit how far this guidance can be taken. The accumulation precision of the hardware is not under the practitioner's control—Hopper retains 13 fraction bits for addition [37]—so a configuration validated on one generation may not transfer to another. And the scheduling of scaling-factor updates, including whether a running history of observed maxima is used to set the factor for the next step, is an implementation choice made by training libraries rather than a matter settled in the works cited here; we, therefore, do not offer a recommendation on it, and note it as a point on which practitioners should consult their framework's documentation rather than the literature.

3.6. Hybrid Parallelism

Hybrid parallelism combines data parallelism (DP), tensor model parallelism (TMP), and/or pipeline model parallelism (PMP) to jointly optimise memory footprint, compute efficiency, and communication overhead—a balance that no single strategy achieves alone [39,40].

Hybrid parallelism takes three primary forms. DP with TMP: Krizhevsky [38] applies DP to a convolutional network's convolutional layers while using TMP for its fully connected layers. DP with PMP: models are divided layer-wise across GPUs whereas two or more replicas train concurrently, as in PipeDream [28,40] and Chimera [41]. All three together—three-dimensional (3D) parallelism—were prefigured by DistBelief [42], which combined concurrent model replicas with distribution of neurons within a layer and partitioning of layers across machines, making it an early DP/model-parallelism hybrid rather than a formalised 3D system. Megatron-LM [26], DeepSpeed [43], Colossal-AI [44], and Merak [45] provide full 3D support with explicitly formalised TMP and PMP components [27].

3.7. Offloading

ZeRO-Offload [43] reduces GPU memory pressure by selectively migrating model states—parameters, gradients, and optimiser state (e.g., Adam momentum and variance)—from GPU VRAM to host CPU memory during training. Because model states are the primary source of memory bottlenecks in large-language-model training, this offloading enables models that would otherwise exceed GPU capacity to be trained on a single or small number of GPUs [43].

The design challenge is choosing which states to offload without making the CPU or PCIe the bottleneck. From a data-flow analysis of the training graph, ZeRO-Offload derives a partition that retains the compute-intensive forward and backward passes on the GPU

while relocating optimiser states and the parameter update to the CPU: only gradients travel GPU→CPU and updated parameters back, all other movement being eliminated [43]. Ren et al. [43] prove this partition optimal within the model they analyse: among strategies that offload optimiser state and the parameter update, no alternative achieves a lower GPU memory footprint without increasing either CPU computation or communication volume [43]. The claim is, therefore, one of optimality under stated assumptions—a fixed two-tier GPU/CPU hierarchy, Adam-style optimiser state, and this cost model—rather than of optimality among all possible memory-reduction schemes; sharding across data-parallel ranks, recomputation and quantisation occupy a different design space and are not ranked by that result. Within those assumptions, it is what allows billion-parameter models to train on a single GPU.

Current limitations of offloading. Despite its advantages, CPU offloading introduces PCIe bandwidth as a hard ceiling on gradient transfer throughput. In practice, the available PCIe bandwidth between host and GPU is orders of magnitude lower than GPU high-bandwidth memory (HBM) bandwidth, meaning that gradient communication becomes the bottleneck as model size or batch size grows [43]. ZeRO-Infinity [46] extends this idea further by leveraging GPU, CPU, and NVMe memory, fitting models whose parameter counts reach the tens—and even hundreds—of trillions for training on current-generation GPU clusters, and fine-tuning models at trillion-parameter scale on one NVIDIA DGX-2 node [46]. Two qualifications belong with that figure. The reported reach is a *memory-capacity* result: it states what can be made to fit, not what can be trained to convergence in an acceptable time, and the cited configurations are DGX-class nodes with NVMe and high-bandwidth interconnect rather than commodity hardware. Because each offloaded tensor crosses PCIe and then a storage interface whose bandwidth is orders of magnitude below HBM, arithmetic intensity per byte moved falls sharply and iteration throughput drops accordingly; the technique converts an infeasible job into a slow one, which is the right trade only when the alternative is not training at all. Fine-grained overlap scheduling between CPU computation and PCIe transfers remains an active area of engineering research [47].

3.8. Activation Checkpointing

Backpropagation requires access to the intermediate activations produced during the forward pass in order to compute gradients. Retaining all of these activations in GPU memory simultaneously constitutes a major memory bottleneck when training large models. Activation checkpointing, introduced by Chen et al. [48], addresses this by storing only a sparse subset of activations-checkpoint tensors and recomputing the remaining ones on demand during the backward pass [4,48].

In a Transformer, only the input tensor to each major block is retained; when the backward pass reaches that block, a localised forward pass regenerates the internal activations from the saved input, which are used immediately and discarded. The technique trades a fixed amount of recomputation for a substantial reduction in peak memory: a 24-layer Transformer with four to five linear sub-layers per layer retains 24 tensors rather than approximately 120, permitting considerably more parameters within the same memory budget [4,48].

3.9. Neural Scaling Laws and Compute-Optimal Training

Empirical research on Large Language Models (LLMs) has demonstrated that model performance improves predictably, not randomly, as computational resources are scaled, following a precise set of relationships known as Neural Scaling Laws [49].

Kaplan et al. [49] established that test loss L scales as a power law in three factors—non-embedding parameters N , training tokens D , and computation C —allowing the per-

formance of massive models to be predicted from smaller runs. Their analysis indicated N was the dominant limiting factor, implying that a fixed budget should prioritise parameters over data [49].

Hoffmann et al. [50] challenged that allocation, arguing many large LLMs were significantly undertrained, and determined the compute-optimal frontier by jointly scaling N and D . Their stated finding is that model size and training-token count should grow in *equal* proportion: each doubling of parameters calls for a doubling of tokens [50]. They demonstrate this by training Chinchilla, a 70B-parameter model, on 1.4 trillion tokens, which uniformly and significantly outperforms the larger but undertrained 280B-parameter Gopher at the same compute budget [50].

The widely quoted figure of “approximately 20 tokens per parameter” does not appear as such in that work; it is arithmetic on the configuration above (70B parameters against 1.4T tokens), and the compute-optimal model in their Table 3 implies a similar figure. We use the ratio in this sense throughout—as a convenient summary of Chinchilla’s own configuration rather than as a law stated by its authors—because the distinction matters for the deviations discussed next: what Hoffmann et al. established is an *equal-scaling* relationship [50], and a fixed ratio is only its consequence under the dense assumptions in which it was derived.

The prescription is, therefore, a shift from maximising model size to balancing model and data scale, which directly raises the data-throughput and memory-management demands on the distributed training systems described above.

Deviations from the Compute-Optimal Frontier

As noted above, the 20-tokens-per-parameter figure is a summary of one compute-optimal configuration rather than a constant, and the equal-scaling relationship it summarises was derived under assumptions that two subsequent lines of work show do not hold generally.

Sparse architectures decouple the two axes. Chinchilla assumes a dense model in which every parameter participates in every forward pass, so N measures parameters and compute simultaneously. Routed models break this identification. Clark et al. [51], across models with hundreds of experts spanning five orders of magnitude in size, find that parameter count and compute requirement behave as two separate axes, each improving performance when increased, and collapse the two into an Effective Parameter Count on which all models share a single scaling rate. The consequence is structural: once the axes separate, “model size” is no longer a single quantity a token ratio can be stated against. DeepSeek-V3 shows the magnitude—671B total parameters, 37B activated per token, 14.8T tokens [36]: roughly 22 tokens per *total* parameter but roughly 400 per *activated* parameter. Which the Chinchilla rule constrains is precisely what the dense formulation cannot answer.

Repetition bounds the data axis. Scaling D with N presumes additional unique tokens exist; for most languages and domains, they do not. Muennighoff et al. [52], using up to 900 billion tokens and models up to 9 billion parameters, find that repeating data for as many as four epochs changes loss negligibly relative to unique data, whereas with further repetition the value of added compute falls away to zero, and propose a scaling law in which repeated tokens and excess parameters progressively lose worth. The D axis thus has a ceiling set by corpus size rather than budget.

Implication for system design. The two deviations pull requirements in opposite directions. Sparsity relieves the compute axis while inflating the memory and communication requirements of Sections 3.6 and 3.7, since total parameters must still be stored and routed. Data constraints cap returns on precisely the high-throughput pipeline dense scaling laws

demand. A configuration derived from the Chinchilla ratio alone mis-provisions in both dimensions for any model that is sparse or data-limited.

Further caveats. Three limitations apply beyond the deviations above. The laws measure loss on next-token prediction and do not directly predict downstream performance on reasoning, coding, or instruction-following benchmarks, which may scale differently. The Chinchilla analysis [50] used a fixed set of architectures, so changes to the attention mechanism or data mixture can shift the optimal N -to- D ratio. Additionally, they ignore inference cost: a smaller well-trained model may be preferable in deployment even where a larger one achieves lower training loss at equal compute. Scaling laws are, therefore, heuristics, not prescriptions [49].

4. Text Generation and Decoding Strategies

4.1. The Challenge of Neural Text Degeneration

A critical challenge in autoregressive generation is the phenomenon of **neural text degeneration**—repetitive or incoherent sequences produced despite high per-token confidence—which arises because maximisation-based decoding systematically diverges from the statistical properties of human text. Holtzman et al. [53] analysed why traditional methods such as Greedy Search and Beam Search fail in open-ended generation tasks and proposed nucleus sampling as a solution.

Three findings from Holtzman et al. [53] account for the failure. High-quality human text does not occupy the highest-probability regions of the model’s distribution: natural language routinely assigns mass to tokens that are not the single most likely continuation, maintaining information density and stylistic variety. Beam Search consequently falls into repetitive loops, the model becoming over-confident in a sequence from which no stochasticity permits escape. Maximising cumulative log-probability yields text that is safe, predictable, and monotonous, degrading further over long passages as context is lost [53].

4.2. Sampling-Based Decoding Results

Nucleus (Top- p) sampling [53] responds by making the candidate pool adaptive. Unlike Top- k , which fixes the pool size regardless of distribution shape, Top- p sums token probabilities in descending order and draws from the minimal set whose cumulative mass reaches p , so the effective vocabulary expands when the model is uncertain and contracts when it is confident. The reported result is text matching the lexical diversity and statistical properties of human language far more closely than Beam Search, mitigating degeneration and enabling coherent long-form generation that remains diverse without sacrificing fluency [53].

4.3. Beyond Nucleus Sampling

Nucleus sampling addresses *what* to sample from. Two subsequent lines of work address what it leaves open: obtaining the quality benefit without a stochastic search, and reducing the cost of autoregressive generation itself.

Contrastive decoding: quality without sampling. Li et al. [54] share Holtzman et al.’s diagnosis—maximum probability serves open-ended generation poorly, yielding short, repetitive output, while sampling risks incoherence—but change the objective rather than inject randomness. Contrastive decoding optimises a contrastive objective subject to a plausibility constraint: the gap in likelihood between a large “expert” model and a smaller “amateur” one [54]. Failure modes common to both—generic continuations, repetition—are suppressed by the subtraction, whereas the expert’s advantages survive. It reports higher quality than the larger model alone, significantly outperforming four strong decoders, in-

cluding nucleus and top- k across Wikipedia, news, and story domains, with zero additional training [54], at the cost of a second forward pass per step.

Speculative decoding: cost without a quality trade-off. Leviathan et al. [55] attack the sequential bottleneck instead: a lightweight draft model proposes candidate tokens which the large model verifies in parallel, with rejection sampling determining acceptance. Uniquely among the methods here, it involves no quality trade-off—it operates without changing the output distribution, producing identical outputs to the original model, at a $2\times\text{--}3\times$ acceleration over the standard T5X implementation [55].

That guarantee is why it composes with the others rather than competing: verification preserves the target distribution exactly, so speculative decoding layers beneath nucleus sampling without altering the text that scheme would produce. Decoding choice is, therefore, not a single axis. Nucleus and contrastive decoding trade quality characteristics against compute; speculative decoding trades memory—a second resident model—for latency at fixed quality; constrained decoding, restricting generation to a formal grammar or schema, imposes validity rather than optimising quality. Willard and Louf reformulate that restriction as transitions between the states of a finite-state machine, which permits an index over the vocabulary to be constructed in advance so that generation can be guided by regular expressions or context-free grammars while adding little overhead to token generation [56]. The balance is thus separable into an objective decision, a sampling decision, and an execution-schedule decision, made largely independently.

5. Adaptation of Large Language Models for Specific Tasks

Adapting a pre-trained LLM requires choosing between two strategies with different cost profiles. **Fine-Tuning** updates parameters on task-specific data, capturing deep task knowledge and generalising within the domain, but incurring compute cost, requiring labelled examples, and being unable to incorporate information post-dating the run. **Retrieval-Augmented Generation (RAG)** retrieves documents at inference and injects them as context, requiring no weight updates and supporting real-time refresh, but remaining sensitive to retrieval quality and adding latency proportional to index size. The following sections detail both and identify when each is preferable.

5.1. Fine-Tuning

Fine-tuning is a transfer learning technique in deep learning whereby a large neural network, previously pretrained on a massive, general-purpose dataset, undergoes additional supervised training on a smaller, task-specific or domain-specific dataset [57].

Table 3 contrasts the two phases. The dimensions are not independent: each row follows from the first. Because pre-training is self-supervised on unlabelled corpora, it can consume data at a scale fine-tuning cannot, which is what makes it computationally intensive (weeks to months) and what obliges it to update the entire model. Because fine-tuning targets a specific task, it requires labelled data and that is precisely why updating a parameter subset suffices and why days-to-weeks costs are achievable [57]. The trade-off the table implies is, therefore, not a choice between two methods but a dependency: fine-tuning is cheap only because pre-training has already been paid for, and the parameter-efficient methods of Section Comparison of PEFT Methods extend that logic one step further.

Table 3. Comparison of Pre-training and Fine-tuning. Adapted from [57].

Aspect	Pre-Training	Fine-Tuning
Definition	Training on a vast amount of unlabelled text data	Adapting a pre-trained model to specific tasks
Data requirement	Extensive and diverse unlabelled text data	Smaller, task-specific labelled data
Objective	Build general linguistic knowledge	Specialise model for specific tasks
Process	Data collection, training on large dataset, predict next word/sequence	Task-specific data collection, modify last layer for task, train on new dataset, generate output based on tasks
Model modification	Entire model trained	Last layers adapted for new task
Computational cost	High (large dataset, complex model)	Lower (smaller dataset, fine-tuning layers)
Training duration	Weeks to months	Days to weeks
Purpose	General language understanding	Task-specific performance improvement
Examples	GPT, LLaMA 3	Fine-tuning LLaMA 3 for summarisation

Three fine-tuning paradigms trade data availability against directional signal [57]. **Un-supervised fine-tuning** continues training on unannotated domain text, enriching domain-specific representation without labels—useful in specialised fields such as biomedicine or law, but providing weak signal for tasks requiring structured outputs. **Supervised fine-tuning (SFT)** optimises on labelled input-output pairs, suiting classification, sentiment analysis, or domain-specific question answering, at the cost of collecting and curating high-quality annotations. **Instruction-based fine-tuning** replaces labelled examples with natural-language instructions, reducing dependence on annotated corpora and suiting conversational assistants, but becoming sensitive to instruction quality [57].

Parameter-Efficient Fine-Tuning (PEFT). Full fine-tuning, in which all model parameters are updated, is computationally prohibitive for models with billions of parameters. Parameter-Efficient Fine-Tuning (PEFT) methods address this by updating only a small, targeted subset of parameters while keeping the majority of the pre-trained weights frozen, thereby drastically reducing memory and compute requirements [57].

The most widely adopted PEFT technique is **Low-Rank Adaptation (LoRA)** [58], which injects trainable low-rank matrices into selected Transformer weight matrices while keeping the original weights frozen. Specifically, for a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$, the weight update is factorised as $\Delta \mathbf{W} = \mathbf{B}\mathbf{A}$, where $\mathbf{A} \in \mathbb{R}^{r \times k}$ and $\mathbf{B} \in \mathbb{R}^{d \times r}$ with rank $r \ll \min(d, k)$. Only $\mathbf{A}$ and $\mathbf{B}$ are updated during fine-tuning, at inference the product $\mathbf{B}\mathbf{A}$ is merged back into $\mathbf{W}$, leaving inference-time compute unchanged relative to the original model [58]. A further extension, **QLoRA** [35], combines LoRA with 4-bit quantisation of the frozen base model, enabling fine-tuning of 65B-parameter models on a single 48 GB GPU while preserving near full fine-tuning quality [35]. Other PEFT approaches include **Prefix Tuning** [59] and **Prompt Tuning** [60], which prepend learnable continuous tokens to the input or intermediate layers rather than modifying weight matrices, and **Adapter layers** [61], which insert small bottleneck modules between Transformer blocks.

PEFT methods have largely superseded full fine-tuning for domain adaptation in resource-constrained settings, as they achieve competitive task performance while training on fewer parameters in many configurations [35,57,58].

Comparison of PEFT Methods

Table 4 places the principal PEFT methods side by side. One caveat governs how it is read: the figures are those reported by each originating publication and are *not measured against a common baseline*—LoRA’s against GPT-3 175B with Adam, the adapter figures against BERT on GLUE, prefix-tuning’s against GPT-2 and BART on generation. The param-

eter fractions are, therefore, not directly comparable; the table summarises each method's design point rather than a controlled comparison. To our knowledge, no evaluation of all five under a single model, task suite, and budget exists, and establishing one would be a useful contribution.

Table 4. Comparison of parameter-efficient fine-tuning methods. Figures are as reported by each originating publication and are measured against *different* base models and task suites; they are, therefore, indicative of each method's design point rather than directly comparable. See the caveat in Section Comparison of PEFT Methods.

Method	Mechanism	Reported Efficiency	Reported Performance	Inference Cost
LoRA [58]	Trainable low-rank matrices BA injected into frozen weights	10,000× fewer trainable parameters and 3× less GPU memory vs. GPT-3 175B with Adam	Comparable to full fine-tuning	None; BA merges into W, so no additional inference latency
QLoRA [35]	LoRA over a frozen base quantised to 4-bit NormalFloat (NF4); double quantisation; paged optimisers	65B model finetuned on a single 48 GB GPU	Preserves full 16-bit fine-tuning task performance; Guanaco reaches 99.3% of ChatGPT on the Vicuna benchmark after 24 h on one GPU	Dequantisation overhead on the frozen base
Adapters [61]	Small bottleneck modules inserted between Transformer blocks	3.6% additional parameters per task	Within 0.4% of full fine-tuning on GLUE	Additional latency from the inserted modules [58]
Prefix Tuning [59]	Learnable continuous prefix prepended to activations; weights frozen	0.1% of parameters	Comparable in the full-data setting; outperforms fine-tuning in low-data settings; extrapolates better to unseen topics	Prefix occupies context positions
Prompt Tuning [60]	Learnable soft prompt prepended to the input	Task-specific parameters only	Closes the gap with full model tuning as models exceed billions of parameters; more robust to domain transfer than full tuning	Prompt occupies context positions

Read with that caveat, the table supports three conclusions the individual papers do not state in isolation.

First, the methods differ in *where* they spend the adaptation budget, and this governs inference behaviour more than parameter count does. LoRA and QLoRA modify the weight space, so the adaptation folds back into **W** and costs nothing at inference [58]. Adapters modify the computation graph, so their cost is permanent—a contrast the LoRA authors draw explicitly [58]. Prefix and prompt tuning modify the input space, so their cost is neither parameters nor latency but *context*: the learned prefix consumes positions in a window whose quadratic cost Section 2.2 established, making them least attractive precisely where context is scarcest.

Second, the trade-offs are scale-dependent. Prompt tuning closes the gap with full model tuning only as models exceed billions of parameters [60], whereas prefix tuning

reports its largest advantage in the opposite regime, outperforming fine-tuning in low-data settings [59]. These describe different axes; the choice turns on data availability and model scale, not a single efficiency ranking.

Third, QLoRA is orthogonal rather than incremental to LoRA: it does not improve the adaptation mechanism but changes the memory class of the frozen base beneath it, which is what moves a 65B model onto a single 48 GB GPU [35]. This is the principle of Section 3.5 again—precision reduced where values are stored and read, preserved where they are updated—and its result is a memory saving, not an accuracy improvement.

5.2. Retrieval Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) [62] addresses the static-knowledge limitation of pre-trained models by querying an external document store at inference time and conditioning generation on the retrieved passages. Lewis et al. [62] demonstrated measurable accuracy gains on fact-seeking question-answering tasks where parametric knowledge alone was insufficient—a class of tasks where correct answers depend on precise factual details that are not reliably memorised during pre-training. In a RAG pipeline, current or proprietary information is retrieved at inference time rather than encoded in model weights, allowing organisations to adapt any pre-trained LLM to their domain without incurring the costs of full fine-tuning [57,62]. A key failure mode is retrieval quality: if the index returns stale, irrelevant, or contradictory passages, the generator can produce confidently incorrect output, making retrieval precision as critical as the generative model itself.

A typical RAG pipeline proceeds through four stages [57,63]. In **indexing**, source documents are chunked and stored in a searchable database, usually a dense vector index. At query time the input is **processed**—optionally rewritten, or encoded as a dense vector—for compatibility with that index. **Retrieval** then scores and ranks candidate passages: the original RAG system [62] uses Dense Passage Retrieval (DPR), encoding queries and passages into a shared embedding space and selecting by maximum inner-product search, which hybrid systems may supplement with sparse lexical methods such as BM25 to balance recall and precision. Finally, **prompt augmentation** concatenates the top-ranked passages with the query, supplying grounded context for generation [57,63].

The advantages follow directly: responses can incorporate information post-dating the training cutoff, factual accuracy improves through grounding in verifiable sources, and proprietary knowledge is integrated without modifying weights—a computationally inexpensive path to domain adaptation relative to fine-tuning [57,62]. Production deployment introduces corresponding costs: sensitivity to retrieval latency, cost management at query volume, index freshness as source data evolves, and data-security and governance requirements in regulated environments [57]. Typical deployments include document-grounded question answering for customer support, enterprise search augmenting keyword results with generated summaries, and internal knowledge systems covering HR policy or compliance documentation [57].

Beyond the Four-Stage Pipeline

The four-stage pipeline describes what Gao et al. [63] term Naive RAG, whose progression to Advanced and Modular RAG their survey traces across the tripartite foundation of retrieval, generation, and augmentation. Each stage embeds an assumption, and each subsequent technique addresses a specific failure of one rather than improving the pipeline generically.

The chunking assumption: hierarchical retrieval. Indexing assumes a document can be cut into independent chunks without loss. Sarthi et al. [64] identify the failure: existing methods retrieve only short contiguous chunks, limiting holistic understanding of document context.

A query requiring synthesis of a whole document cannot be served by any single chunk, so no ranking improvement retrieves it. RAPTOR changes what is indexed rather than how it is ranked—recursively embedding, clustering, and summarising chunks into a tree with differing levels of summarisation, retrieving across abstraction levels—and reports 20% absolute accuracy over the previous best on QuALITY with GPT-4 [64], indicating that flat chunking, not retrieval quality, was the binding constraint.

That conclusion is now supported by controlled evidence rather than a single system comparison. Shaukat et al. [65] benchmark 36 segmentation methods—fixed-size, semantic, structure-aware, hierarchical, adaptive and LLM-assisted—across six knowledge domains and five embedding models, scoring retrieval with normalised DCG@5 alongside Hit@5 and MRR. Content-aware segmentation improves retrieval effectiveness substantially over naive fixed-length splitting: their best strategy, paragraph-group chunking, reaches a mean nDCG@5 of approximately 0.459 with Precision@1 near 24%, whereas fixed-size character baselines fall below 0.244 nDCG@5 at 2–3% Precision@1 [65]. Three findings bear directly on the argument of this section. The best strategy is domain-dependent—dynamic token sizing leads in biology, physics and health; paragraph grouping in legal and mathematical text—so chunking is a per-corpus engineering decision rather than a default. Larger embedding models raise absolute scores but remain sensitive to poor segmentation, so capacity does not compensate for a bad index. And the gains are not free: producing more and smaller chunks increases index size and query latency [65], which is the same storage-against-latency trade that Section 3 identifies at the hardware level, reappearing at the retrieval layer.

Two further choices govern retrieval quality and are, in our reading, under-reported relative to their influence. The first is negative sampling: a dense retriever is trained to separate relevant from irrelevant passages, so the difficulty of the negatives it is shown during training determines whether it learns a discriminative boundary or a superficial one, and hard negatives mined from the corpus itself behave differently from randomly drawn ones. The second is evaluation. Retrieval is commonly reported with rank-sensitive measures—nDCG, Hit@*k* and MRR are the three used in the benchmark above [65]—and the choice among them is not neutral, because a system tuned for Hit@5 need not be the system that maximises nDCG@5 when the generator consumes only the top one or two passages. We flag both as decisions a deployment must make explicitly, and note that neither is settled by the retrieval literature in a way that would let us state a default here.

The single-vector assumption: late interaction. Dense retrieval compresses a passage into one embedding, forcing every query to be served by a single point. Both extremes disappoint: a cross-encoder reranker is accurate but does not scale, a single-vector index scales but discards token-level detail. ColBERT [66] defers the interaction, independently encoding query and document then applying a cheap interaction step modelling fine-grained similarity, retaining per-token representations while remaining indexable—two orders of magnitude faster than BERT-based rankers at four orders of magnitude fewer FLOPs per query [66]. Hybrid sparse–dense retrieval, supplementing the dense index with BM25, is a coarser instance of the same principle: recovering the exact-match signal a dense embedding discards.

The static-retrieval assumption: self-reflection. The naive pipeline retrieves once and trusts the result. Both halves fail: retrieval is wasted when parametric knowledge suffices, and harmful when the index returns irrelevant passages the generator cannot reject. Self-RAG [67] makes both decisions learnable via reflection tokens, adaptively retrieving on demand and reflecting on retrieved passages while remaining controllable at inference; it reports results above ChatGPT and retrieval-augmented Llama2-chat across open-domain QA, reasoning, and fact-verification tasks, with significant gains in factuality and citation

accuracy for long-form generation [67]. This is the most direct published answer to the hallucination-suppression problem of Section 7.

What remains unsolved. The costs compound: hierarchical indexing multiplies construction cost and must be rebuilt as data changes, late interaction multiplies index size, self-reflection multiplies inference cost. None resolve conflict between passages individually well ranked but mutually contradictory, nor establishes which source to trust. Multi-turn RAG compounds all of these, since the query itself becomes an object of inference. The Naive-to-Modular trajectory [63] is, therefore, less a sequence of improvements than a progressive relaxation of the assumptions that made the original pipeline tractable to describe.

The context-budget assumption: compression. Each technique above increases the number of tokens placed before the generator, and the context window is finite and quadratically costly (Section 2.2). Prompt compression attacks that budget directly rather than improving what is retrieved. Jiang et al. [68] propose a coarse-to-fine scheme combining a budget controller that maintains semantic integrity under high compression ratios with a token-level iterative algorithm that models the interdependence between retained fragments, reporting up to $20\times$ compression with little performance loss across mathematical reasoning, instruction-following and conversational datasets [68]. The consequence for a retrieval pipeline is that retrieval quality and context economy are separable levers: a system may retrieve more aggressively provided that what it forwards is compressed, and the cost of a better retriever can be paid in tokens saved elsewhere.

A false dichotomy: retrieval and fine-tuning together. This survey presents fine-tuning and retrieval as answers to different questions, and Table 3 contrasts them, but they are not exclusive and in deployment are frequently combined. The two change different things: fine-tuning alters the weights, and therefore, the model's behaviour, output format and domain idiom, whereas retrieval alters the context, and therefore, the facts available at inference. A system required both to follow a house style and to cite current documents needs both, and the common production pattern is a parameter-efficiently adapted model serving as the generator inside a retrieval pipeline [57,62]. Treating them as alternatives is an artefact of comparing them on a single axis—the cost of incorporating new information—on which they genuinely do compete.

5.3. Reinforcement Learning from Human Feedback (RLHF)

Fine-tuning and RAG improve task performance and knowledge grounding, but neither of them guarantees that model outputs align with safety requirements. Reinforcement Learning from Human Feedback (RLHF) [69] addresses this by framing alignment as a reward-optimisation problem.

The RLHF pipeline proceeds in three stages. First, supervised fine-tuning (SFT) on a curated set of demonstration dialogues produces an instruction-following starting point. Second, a reward model is trained on human-ranked pairs of model responses, learning a scalar score that captures annotator preferences for helpfulness, harmlessness, and honesty. Third, the SFT-initialised policy is optimised by the reward model using Proximal Policy Optimisation (PPO), subject to a KL-divergence penalty that constrains deviation from the SFT baseline and thereby prevents reward hacking.

Ouyang et al. [69] report that outputs from the 1.3B-parameter InstructGPT model are preferred by human evaluators to outputs from the 175B-parameter GPT-3, despite having $100\times$ fewer parameters—establishing that alignment quality is an axis orthogonal to raw scale. RLHF has since been adopted as a standard post-training step across subsequent instruction-tuned systems [57,69].

Current limitations include the high cost of human preference annotation, reward-model brittleness when the policy over-optimises a proxy objective, and limited generalisation across domains [57,69].

5.3.1. Reward Hacking and the Limits of a Proxy Objective

The KL penalty is often presented as a regularisation detail. It is better understood as a response to a failure structural to the RLHF formulation.

The reward model is not the objective but a learned proxy for it. Gao et al. [70] find that pushing a proxy's value too high degrades ground-truth performance—the pattern Goodhart's law predicts—and—their central result—that the divergence is predictable rather than erratic: it follows scaling laws whose functional form depends on the optimisation method, differing between reinforcement learning and best-of- n sampling, and whose coefficients scale smoothly with reward-model parameter count, studied as a function of the KL-penalty coefficient [70].

Three consequences follow. First, reward hacking is a budget to be spent rather than a bug to be fixed: since the proxy diverges predictably under optimisation pressure, the KL coefficient tunes how far the policy may improve on the proxy against how far it drifts from what the proxy approximates. Second, because the coefficients scale with reward-model size, a larger policy trained against a fixed reward model degrades sooner, not later. Third, the true reward is unobservable during training, so the divergence cannot be detected from the training signal—which is why alignment quality cannot be monitored as a training loss can.

5.3.2. Alternatives to RLHF

RLHF is expensive in two currencies: human preference annotation, and the complexity of running PPO with a reward model and a KL reference model resident simultaneously. The principal alternatives each remove one, and they are not interchangeable.

Direct Preference Optimisation (DPO) removes the reinforcement learning. Rafailov et al. [71] introduce a parameterisation of the RLHF reward model permitting the optimal policy to be extracted in closed form, so the standard RLHF problem is solved with only a simple classification loss; the resulting procedure is stable, performant, and computationally lightweight, eliminating sampling from the language model during fine-tuning and significant hyperparameter tuning, and is substantially simpler to implement than PPO-based RLHF while matching or exceeding it on several benchmarks [71]. It does *not* remove the data requirement: paired preferences are still consumed.

The contrast with PPO is worth stating directly, because the two differ in what is optimised rather than merely in implementation effort. PPO is an *online* procedure: a reward model is first fitted to human preferences, then the policy is improved against that model by repeatedly sampling its own outputs, scoring them, and taking constrained gradient steps, with a KL penalty against a frozen reference policy to prevent drift. Three components are, therefore, resident and interacting during training—policy, reward model, and reference—and the training signal depends on samples the policy itself generates, which is what makes the procedure sensitive to the KL coefficient and to reward-model error. DPO removes the first and third of those roles by algebra rather than by approximation: because the optimal policy under a KL-constrained reward objective can be written in closed form, the reward can be re-expressed in terms of the policy itself, and fitting preferences becomes a classification problem over a *fixed* set of preference pairs [71]. No reward model is trained, nothing is sampled from the policy during optimisation, and the KL constraint is absorbed into the objective rather than tuned as a separate control. What survives the

substitution is the proxy itself—the preference data still stands in for the objective one actually wants—which is why Section 5.3.1 applies to DPO as much as to PPO.

Kahneman–Tversky Optimisation (KTO) removes the pairing requirement. Ethayarajh et al. [72] observe that existing alignment objectives implicitly encode cognitive biases described by prospect theory, such as loss aversion, naming this family human-aware losses (HALOs). Making the Kahneman–Tversky utility model explicit, KTO needs nothing beyond a binary indication that an output is desirable or undesirable, while matching or exceeding preference-based methods at 1B–30B scale [72]. The significance is data-collection economics: binary desirability judgements are cheaper and often already exist as production signals; curated preference pairs do not.

Constitutional AI (CAI) removes the human labels. Bai et al. [73] train a harmless assistant in two stages: a supervised stage of self-critique and revision, then a reinforcement-learning stage in which an AI rather than a human judges which of two samples is preferable, yielding a preference model used as reward—reinforcement learning from AI feedback (RLAIF). The distinguishing claim is the scope of supervision retained: the only human oversight is a list of rules or principles, and the method trains without any human labels identifying harmful outputs [73].

Table 5 shows these are not substitutes for one another: each removes a different cost, and none removes the reward-hacking problem of Section 5.3.1. DPO and KTO relocate the proxy into the loss function rather than eliminating it; CAI replaces a human proxy with an AI one, changing who specifies the objective without making it exact. The 2025 generation takes this trajectory further: DeepSeek-R1 [74] shows that on verifiable tasks—mathematics, coding competitions, STEM—reasoning ability can be incentivised through pure reinforcement learning, obviating human-labelled reasoning trajectories, with patterns such as self-reflection and verification emerging during training rather than being demonstrated, and the resulting model surpasses counterparts trained by conventional supervised learning on human demonstrations [74].

Table 5. Alignment methods compared by the cost each removes. Figures and claims are as reported by each originating publication; as with Table 4, the evaluations are not conducted under a common protocol and the entries are, therefore, not directly comparable.

Method	Supervision Required	Optimisation	Cost Removed	Reported Outcome
RLHF / PPO [69]	Human-ranked response pairs	Reward model + PPO with KL penalty	– (baseline)	Outputs of the 1.3B Instruct-GPT preferred by human evaluators to those of the 175B GPT-3, despite 100× fewer parameters
DPO [71]	Paired preferences (unchanged)	Closed-form optimal policy; simple classification loss	The RL stage and reward model	Matches or exceeds PPO-based RLHF, no LM sampling during fine-tuning, no significant hyperparameter tuning
KTO [72]	Binary desirable/undesirable signal	HALO objective	The pairing requirement in the data	Matches or exceeds preference-based methods at 1B–30B scale
Constitutional AI [73]	A list of rules or principles	Self-critique/revision, then RL from AI feedback (RLAIF)	Human labels identifying harmful outputs	Harmless assistant trained without human harmlessness labels

Over-alignment and the residual risk. Removing annotation cost does not remove the possibility of optimising the wrong thing. The over-optimisation dynamic of Section 5.3.1 attends *any* learned proxy, and each method retains one: DPO’s implicit reward, KTO’s utility model, CAI’s AI preference model. CAI’s proxy is specified by a written constitution, making the objective auditable as a learned reward model is not, but making alignment

contingent on the completeness of a rule list—a different failure surface, not a smaller one. The cost of human feedback is falling; the difficulty of specifying an objective that survives optimisation pressure is not.

6. Practical Configuration Guidance

The preceding sections establish each technique and its constraints. Choosing a training configuration is in practice a sequence of eliminations rather than an optimisation: each constraint removes options until few remain.

6.1. Selecting a Parallelism Configuration

1. **Establish the memory floor.** Weights, gradients, and Adam optimiser states scale with parameter count independently of batch size (Section 3). If this floor exceeds device memory, no batch-size reduction helps and data parallelism alone is eliminated, since each device must still hold the full model [4].
2. **Apply tensor parallelism within the node.** It divides the floor by n but requires two all-reduces forward and two backward per transformer layer [26], on every layer's critical path. Given 300 GB/s intra-server via NVSwitch against 100 GB/s inter-server [26], the tensor-parallel degree is bounded by the node (Section 3.6).
3. **Extend across nodes with pipeline parallelism.** Stages communicate only activations forward and gradients backwards at boundaries [27]—far less volume than per-layer collectives, which is what suits pipelining to the slower link. The cost is the bubble of Section 3.4, mitigated but not eliminated by micro-batching and 1F1B scheduling [27–30].
4. **Replicate with data parallelism for throughput.** Once the model fits, remaining devices scale throughput, synchronising gradients once per step rather than per layer [4].
5. **Offload only when devices are unavailable.** ZeRO-Offload trades PCIe bandwidth for capacity [43] and that ceiling binds as model or batch size grows (Section 3.7). It lowers the hardware barrier; it does not improve throughput.

The ordering reflects one principle: *spend the fastest interconnect on the most frequent communication*. Per-layer collectives get NVLink, per-stage transfers InfiniBand, per-step reductions what remains; PCIe is used only when the alternative is not training at all.

6.2. A Mixed-Precision and Memory Checklist

Table 6 consolidates the settings established in this survey as a starting point that makes the reasoning explicit, not as tuned values: the correct configuration depends on interconnect topology, and every figure is drawn from the source indicated rather than from our own measurements.

Read as a whole rather than row by row, the checklist records a single rule applied at successive levels of the stack: reduce precision, memory residency or replication wherever a value is only stored and read, and preserve it wherever a value is accumulated or updated. The compute-precision, optimiser-state, and precision-exclusion rows are three statements of that rule at decreasing granularity, and the parallelism and activation-memory rows apply the same logic to placement rather than to number format. The consequence for practice is that the entries are not independent switches to be tuned separately: relaxing one of them shifts pressure onto the resource the next row governs, which is why the configuration is best chosen by identifying which constraint binds first at the intended scale rather than by optimising each line in turn.

Table 6. Configuration checklist consolidating the constraints established in this survey. Entries reference the section in which each constraint is derived and the source from which the governing figure is taken. These are reasoning defaults, not tuned recommendations; no configuration here was measured by the authors.

Decision	Governing Constraint	Implication
Compute precision	BF16 preserves the FP32 exponent, avoiding FP16 loss-scaling [32]. FP8 GEMMs are viable at scale but require sub-tensor scaling [36]	Default to BF16. Adopt FP8 only with fine-grained scaling (1×128 activation tiles, 128×128 weight blocks) and expect dequantisation overhead [36,37]
Optimiser state	FP16/FP8 updates underflow; an FP32 master copy is required [31]. DeepSeek-V3 retains master weights, gradients and optimiser states above FP8 [36]	Never quantise the accumulator. Reduce precision where values are <i>stored and read</i> , never where they are <i>accumulated</i>
Precision exclusions	Embedding, output head, MoE gating, normalisation and attention operators retained in BF16/FP32 [36]	Exclude discrete decisions (routing) and accumulation points from low precision; a routing error changes the computation, not merely its accuracy
Normalisation	Pre-LN yields well-behaved gradients at initialisation; Post-LN requires warm-up [23]. RMSNorm reduces running time 7–64% [22]	Pre-LN + RMSNorm decouples the optimisation schedule from the parallelism configuration [19]
Tensor-parallel degree	Four all-reduces per layer per step; bandwidth asymmetry 300 vs. 100 GB/s [26]	Bound by node size, not device count
Activation memory	Checkpointing trades recomputation for peak memory [4,48]	Apply per transformer block once activations, not weights, dominate
Token budget	Model size and tokens scale <i>equally</i> for <i>dense</i> models; Chinchilla’s configuration (70B on 1.4T tokens) implies ≈ 20 tokens per parameter [50]. Sparse models decouple the axes [51]; >4 epochs of repetition yields decaying returns [52]	Apply the $\approx 20:1$ figure only to dense models with sufficient unique data, and treat it as a summary of one configuration rather than a law; see Section Deviations from the Compute-Optimal Frontier
Adaptation method	LoRA merges at inference with no added latency [58]; QLoRA moves a 65B model onto one 48 GB GPU [35]	Choose by where the budget is spent—weights, graph, or context (Section Comparison of PEFT Methods)

6.3. Scale-Dependent Regimes

Configurations differ qualitatively rather than in degree across scale, which is why one recommendation cannot serve all three.

Around 7B parameters. The memory floor is within reach of a single high-memory accelerator in BF16, so parallelism is a throughput decision rather than a feasibility one: data parallelism suffices and tensor parallelism is unnecessary complexity. Adaptation is the operative question, and QLoRA’s fitting of a 65B model on a single 48 GB GPU [35] implies this scale is comfortably tractable for single-device fine-tuning.

Around 70B parameters. The floor exceeds any single device—roughly 700 GB for weights plus Adam states (Section 3)—so tensor parallelism within the node becomes mandatory and the node boundary becomes the primary design constraint. This is the regime the interconnect asymmetry [26] governs.

At trillion-parameter scale. No intra-node combination suffices and 3D parallelism is unavoidable [26,43–45]. Two qualitative changes appear: sparsity becomes attractive because it decouples capacity from compute—DeepSeek-V3 activates 37B of 671B parameters per token [36]—and dense scaling laws cease to specify the token budget (Section Deviations from the Compute-Optimal Frontier). The memory wall [37] is felt directly here, memory demand growing more than 1000% per year against under 50% annual growth in HBM capacity.

These regimes derive from the published figures cited and the constraints established in this survey; they are not the product of our own benchmarking, and no controlled study comparing configurations across all three scales under a common protocol is known to us. They are a reasoning framework—which constraint binds first at each scale—rather than tuned settings.

7. Discussion

7.1. Open Challenges

The challenges below are grouped by tier of origin and graded as engineering problems with a known path, research problems without one, or fundamental limits. The grouping is not presentational: the recurring pattern in this survey is that a defect at a lower tier surfaces as a failure at a higher one, and mitigations applied where the symptom appears cannot resolve a cause lying below it.

The causal structure. Read by tier rather than by row, the table exposes the dependency this survey is organised around. The *hardware* tier sets what the *system* tier can attempt: the memory wall [37] is why offloading exists, so offloading’s PCIe ceiling [43] is a symptom of a hardware trend rather than an independent defect—which is why no scheduling improvement resolves it. The *system* tier sets what the *data* tier can consume: the pipeline bubble caps throughput, hence tokens per unit time, which interacts with the repetition ceiling [52] to determine whether a corpus can be exploited within a budget. The *objective* tier is independent of the three below and is the only one where more compute does not help—reward hacking [70] worsens with optimisation pressure, so a faster training system reaches the failure sooner.

The lower three tiers constrain *how much* can be trained and the objective tier constrains *whether the result is what was intended*; it is the only one the engineering advances surveyed here actively worsen.

Mitigation paths. Listing the challenges is only half of what the causal reading supports. Because each row of Table 7 names a cause rather than a symptom, it also identifies where a remedy has to be applied to have any effect, and the grade fixes what that remedy can achieve: an *engineering* problem admits a path that closes it given effort, a *research* problem admits partial measures whose limits are known, and a *fundamental* limit admits deferral but not removal. Table 8 gives, for each challenge, the most direct measure the literature surveyed here supports, together with what that measure leaves unresolved. The second column is as important as the first: a mitigation whose residual is unstated invites the error the causal grouping is meant to prevent, namely treating a deferral as a solution.

Table 7. Open challenges by tier of origin, with the causal chain from underlying cause to observed deployment failure. Grade: *Engineering* = path known, effort required, *Research* = no established solution, *Fundamental* = arises from the formulation itself.

Tier	Challenge	Causal Chain	Surfaces As	Grade
Hardware	FP8 accumulation precision [37]	13 fraction bits for addition on Hopper → fine-grained scaling required → dequantisation overhead	Throughput below the nominal 2× despite native FP8 support	Engineering
Hardware	Memory wall [37]	Memory demand >1000%/yr vs. HBM capacity <50%/yr → offload to slower tiers	PCIe-bound iteration time [43,46]	Fundamental
System	Pipeline bubble [27,28]	Sequential dependency → idle stages → micro-batching → residual bubble	Utilisation below device count	Research
System	MoE routing (Section 7.2.1)	Data-dependent routing → runtime-determined communication → static partitioning inadequate	Expert-placement imbalance	Research
Data	Repetition ceiling [52]	Finite unique corpus → >4 epochs → value of added compute decays to zero	Compute over-provisioned against available data	Fundamental
Data	Curation and quality [49,50]	Scaling laws specify token <i>count</i> , not token <i>quality</i>	Unpredictable returns on corpus growth	Research
Objective	Reward hacking [70]	Learned proxy → optimisation pressure → proxy diverges from true reward (Goodhart)	Aligned-looking model that fails off-distribution	Fundamental
Objective	RAG grounding [57,63]	Retrieval is a proxy for relevance → no conflict resolution between passages	Confident output from contradictory sources	Research

Table 8. Targeted mitigation path for each challenge of Table 7, with the residual limit each measure leaves in place. Measures are drawn from the techniques surveyed in Sections 3–5; residuals state why the underlying cause persists.

Tier	Challenge	Targeted Mitigation Path	Residual Limit
Hardware	FP8 accumulation precision	Fine-grained scaling (1×128 activation tiles, 128×128 weight blocks), with discrete operations and accumulation points held above FP8 [36,37]	Dequantisation overhead remains until vendors supply configurable accumulation precision and native fine-grained quantisation [37]
Hardware	Memory wall	Offload optimiser state and the parameter update to CPU on the partition proved optimal under its cost model [43], extend to NVMe at larger scale [46], and trade recomputation for peak memory by activation checkpointing [48]	Relocates the bottleneck to the PCIe link rather than removing it [43]; since the capacity gap is a growth-rate mismatch, each measure buys scale once [37]
System	Pipeline bubble	Micro-batching at $M \geq 4D$, the point at which GPipe reports the bubble negligible [30], combined with 1F1B and interleaved stage assignment [27–30]	In-flight micro-batches raise activation memory, returning the cost to the hardware tier; removing the bubble at scale without prohibitive communication or memory overhead is open
System	MoE routing	Auxiliary-loss-free load balancing [36], with gating kept in high precision so that a quantisation error cannot alter which expert is selected [36]	Communication volume is still fixed at runtime by the router, so the statically partitioned hybrid schemes of Section 3.6 remain a poor fit
Data	Repetition ceiling	Spend the budget on repetition before collection: up to four epochs changes loss negligibly against unique data, and augmenting with code data or relaxing standard filters extends the usable corpus [52]	Past that point, the value of added compute decays to zero [52]; the ceiling is deferred rather than raised, and MoE relieves compute, not the token supply
Data	Curation and quality	Treat filtering as a tunable rather than a fixed pipeline, since removing commonly used filters can help in the data-constrained regime [52]	No scaling law predicts return on quality; token <i>count</i> remains the only quantity the laws specify [49,50]
Objective	Reward hacking	Use the KL coefficient as the explicit budget for how far the policy may exploit the proxy [70], or remove the learned reward model entirely by solving the same objective in closed form with DPO [71]	KL trades divergence against gain rather than eliminating it, and the true reward is unobservable during training, so the failure cannot be read off the loss [70]
Objective	RAG grounding	Retain token-level evidence at indexable cost through late interaction [66], and recover exact-match signal by supplementing the dense index with a sparse one	Improves which passages are retrieved, not what is conducted when they disagree; conflict resolution between contradictory sources remains unaddressed [63]

Read against Table 7, two properties of the mitigation set are worth stating. First, every effective measure acts at the tier of the cause rather than the tier of the symptom: activation checkpointing and offloading answer the memory wall at the hardware tier, and no scheduling improvement at the system tier substitutes for them. Second, the two *fundamental* rows behave differently from the rest—the repetition ceiling and reward hacking admit measures that buy time or bound the damage, but neither admits a path that closes the problem, which is what distinguishes them from the engineering rows above. The practical consequence for a deployment is that effort spent at the wrong tier is not merely inefficient, it is ineffective.

Two challenges in Table 7 warrant comment beyond their tabulated form, because neither reduces to a single causal chain.

Model compression: pruning and sparsity. The training techniques surveyed here produce dense models in which most parameters are non-zero. Post-training compression via weight pruning offers a complementary path to deployment efficiency: structured pruning removes entire attention heads or feed-forward neurons and so remains compatible with standard hardware, whereas unstructured pruning achieves higher compression ratios at the cost of requiring sparse arithmetic support. Post-training methods have advanced correspondingly: SparseGPT prunes GPT-family models to at least 50% sparsity in one shot, without retraining and at minimal accuracy loss [75], whereas Wanda attains comparable results from a far simpler criterion, removing weights with the smallest magnitude multiplied by the corresponding input activation on a per-output basis, and requires neither

retraining nor a weight update [76]. Methods that learn sparse masks during pre-training rather than post hoc are an active frontier, and integrating them with the parallelism and precision techniques described here remains open [77].

Alignment, safety, and ethics. Beyond the reward-hacking dynamic of Section 5.3.1, LLMs raise ethical concerns that no engineering advance addresses: training corpora encode societal biases that can surface as discriminatory outputs, privacy risks arise from models memorising personally identifiable information in training data; and large-scale reproduction of copyrighted text during pre-training leaves unresolved legal questions. These dimensions underscore that the advances surveyed here must develop alongside governance frameworks [2,4,57].

7.2. Current Trends and Future Directions

Three directions are treated in dedicated subsections below rather than listed, because each interacts directly with the parallelism, memory, and precision constraints developed earlier in this survey rather than merely succeeding them.

7.2.1. Mixture-of-Experts Architectures

Mixture-of-Experts is the principal architectural response to the coupling of capacity to compute in the dense scaling of Section 3.9. Proposed by Shazeer et al. [78], MoE routes each token to a learned subset of expert sub-networks rather than the full parameter set.

The decoupling, quantified. Mixtral 8x7B makes this concrete: each layer holds 8 feedforward blocks (experts), and at each layer a routing network picks two of them per token to process the state, combining their outputs—so each token accesses 47B parameters but uses only 13B active during inference, while matching or beating Llama 2 70B and GPT-3.5 on every benchmark evaluated [79]. DeepSeek-V3 applies the principle far more aggressively: 671B total, 37B activated per token [36]. Capacity-to-compute is thus a design parameter, not a constant—the property that invalidates the dense Chinchilla formulation for these models (Section Deviations from the Compute-Optimal Frontier).

Why this is a systems problem. Sparse activation reduces compute per token but not the parameters that must be stored, and it adds a communication pattern dense models lack. All 47B of Mixtral’s parameters must be resident somewhere, though only 13B are used, so MoE relieves the constraint Section 3 identifies as *not* binding (compute) while aggravating the one that is (memory). Because routing is data-dependent, the experts a token needs may not sit on the device holding it, making expert placement a partitioning problem whose communication volume is fixed at runtime by the router rather than statically by the configuration. This is why MoE composes awkwardly with the schemes of Section 3.6, which partition a graph known in advance. It is also why DeepSeek-V3 keeps MoE gating in high precision while moving GEMMs to FP8 [36]: routing decisions are discrete, and a quantisation error that changes which expert is selected does not degrade the output slightly—it changes the computation performed. The separation of total from activated parameters persists in 2025-generation designs: Qwen3’s flagship MoE model activates 22B of its 235B total parameters per token, selecting 8 of 128 experts per layer [80], so the memory-versus-compute asymmetry above remains the governing constraint.

7.2.2. Long-Context Modelling

Section 2.2 established attention’s $O(n^2 \cdot d)$ per-layer cost. Long-context work is best organised by which factor of that expression a method attacks, since the approaches are not alternatives.

Attacking the constant, not the exponent. FlashAttention [81] is an IO-aware *exact* algorithm using tiling to reduce memory reads and writes between GPU HBM and on-chip

SRAM, reporting 15% end-to-end speedup on BERT-large at length 512, $3\times$ on GPT-2 at 1K, and $2.4\times$ on long-range arena at 1K–4K [81]. The essential and frequently misstated point is that it does not change the asymptotic complexity, which remains quadratic; it removes memory traffic, not operations, and its block-sparse extension is separately identified as approximate [81]. The practical limit on context was thus set by the memory hierarchy rather than arithmetic—the bandwidth-bound behaviour recurring throughout Section 3.

Attacking the cache. A third target is neither the operation count nor the memory traffic of the attention kernel, but the state that autoregressive decoding must retain. Because each generated token attends to every previous one, the key and value tensors of all preceding positions are held and re-read at every step, so the cache grows linearly with context and is re-loaded once per token. Shazeer identifies this repeated loading of the large key and value tensors as the memory-bandwidth cost that makes incremental inference slow, and proposes multi-query attention, which shares a single key-value head across all query heads, greatly reducing the size of those tensors and reporting much faster decoding at only minor quality degradation [82]. Ainslie et al. observe that this can cost quality and that training a separate model for inference alone may be undesirable; grouped-query attention instead uses an intermediate number of key-value heads, and uptraining an existing multi-head checkpoint with 5% of the original pre-training compute yields quality close to multi-head attention at a speed comparable to multi-query attention [83]. The significance for this survey is that the constraint is again bandwidth rather than arithmetic: the cache changes no FLOP count, and the architectural response is to shrink what must be re-read per token.

Attacking the operations. Native Sparse Attention [84] instead drops computation, coupling coarse-grained compression of tokens (retaining global context) with fine-grained selection of tokens (retaining local precision) in a dynamic hierarchical strategy aligned to modern hardware, and makes the sparsity natively trainable end-to-end rather than imposed after pre-training; it reports maintaining or exceeding full-attention models across general, long-context, and reasoning benchmarks, with substantial speedups against full attention at 64k sequence length in decoding, forward, and backward passes [84].

Attacking the position representation. A model must also represent positions it was not trained on. Rotary Position Embedding [14] encodes absolute position with a rotation matrix while incorporating relative position dependence into self-attention, and underpins most subsequent context-extension methods.

Attacking the premise. The retrieval methods of Section Beyond the Four-Stage Pipeline select what enters the window rather than extending it. Hierarchical retrieval [64] and long-context attention address the same requirement by opposite means, and the conditions under which one dominates are not established. Section 2.1.3 adds a variable rarely considered with either: the tokeniser determines how much text a window of n tokens holds, varying by up to a factor of 15 across languages [13].

7.2.3. Inference-Time Efficiency

The techniques above reduce the cost of *producing* a model. A deployed model is dominated by a different cost, and the same memory-bandwidth constraint governs it. Three families complete the lifecycle.

Managing the cache. Section 7.2.2 identified the key-value cache as the object that must be re-read per generated token. In a serving system, it is also the object that limits how many requests can be batched, and Kwon et al. [85] observe that its size grows and shrinks dynamically per request, so that naive allocation wastes memory through fragmentation and duplication and thereby caps the batch size. PagedAttention borrows the classical virtual-memory abstraction: the cache is stored in fixed-size blocks that need not be contiguous.

ous, which permits near-zero waste and flexible sharing of cache within and across requests. The system built on it reports $2\text{--}4\times$ the throughput of prior serving systems at comparable latency [85]. The result is notable here because it is a pure memory-management gain: no operation is made faster and no approximation is introduced.

Reducing what must be read. Post-training quantisation compresses weights after training rather than changing the training arithmetic of Section 3.5. GPTQ quantises to 3 or 4 bits per weight in one shot using approximate second-order information, reporting 175B-parameter models quantised in about four GPU hours with negligible accuracy degradation against the uncompressed baseline [86]. AWQ starts from the observation that weights are not equally important and that the salient channels are identified by the *activation* distribution rather than by weight magnitude: protecting roughly 1% of channels, by scaling them rather than by mixing precisions, substantially reduces quantisation error while remaining hardware-friendly, and the accompanying inference implementation reports more than $3\times$ speedup over an FP16 baseline [87]. Both are weight-only methods aimed at the same quantity FlashAttention and PagedAttention target from other directions: bytes moved per token.

Reducing how many steps are taken. Speculative decoding, treated in Section 4.3, belongs to this group: it spends memory on a draft model to convert sequential steps into verified parallel ones without altering the output distribution.

Read together, these confirm the survey's organising claim at the deployment layer. The cache is a bandwidth object, quantisation reduces bandwidth per parameter, and speculative decoding amortises bandwidth across tokens; none of the three principally reduces arithmetic.

7.2.4. Hardware-Aware Training and Co-Design

Section 3's techniques treat hardware as fixed and adapt software to it. The emerging direction inverts this, treating architecture and hardware as a joint design problem.

The clearest published statement is the architectural analysis accompanying DeepSeek-V3 [37], which reports training at frontier quality on 2,048 H800 GPUs and argues that effective software–hardware co-design enables cost-efficient training of large models. It quantifies the driving pressure: memory demand increases by more than 1000% per year against typically under 50% annual growth in high-speed memory capacity such as HBM [37]—the AI memory wall, and the reason memory rather than arithmetic is this survey's recurring constraint. Its recommendations to vendors are specific: configurable accumulation precision, native fine-grained quantisation in Tensor Cores, and dedicated co-processors for communication management [37].

Read against Section FP8 Training at Scale: A Case Study, each recommendation names a place where a software technique surveyed here compensates for a hardware primitive that does not exist: fine-grained scaling for fixed accumulation precision, the parallelism strategies of Section 3.6 for absent communication offload. A complementary line automates parallelism selection given model size and cluster topology, of which Merak's automated 3D parallelism [45] is representative, reducing required expert knowledge but operating within hardware constraints rather than relaxing them.

These directions intertwine training optimisation, architecture design, and inference efficiency: MoE turns an architectural choice into a communication problem, long-context work turns a modelling requirement into a memory-hierarchy problem, and co-design turns both into a hardware procurement problem.

8. Conclusions

This survey traced the engineering dependencies between LLM architecture, distributed training, and adaptation rather than cataloguing each in turn. Three findings follow, none visible when the layers are treated separately.

The binding constraint is memory bandwidth, and it is the same at every layer. Across otherwise unrelated sections, the limiting resource is not arithmetic but the movement of data. Tensor parallelism is bounded by the interconnect rather than device count, its collectives sitting on every layer's critical path against a 300 versus 100 GB/s asymmetry [26]. FlashAttention's speedups come from reducing HBM traffic while leaving the quadratic operation count intact [81]. Offloading converts a memory shortage into a PCIe bottleneck [43]. RMSNorm's saving is bandwidth, not FLOPs [22]. DeepSeek-V3's hardware analysis states the unifying trend: memory demand grows more than 1000% per year against under 50% annual growth in HBM capacity [37]. **Techniques that appear to address different problems—attention implementation, normalisation, parallelism topology, precision format—are responses to one scarcity, which is why they compose well and why optimising any in isolation yields less than expected.**

Each layer's choices are denominated in the next layer's currency. A vocabulary decision taken before training fixes sequence length and thus, by the $O(n^2 \cdot d)$ cost of Section 2.2, an attention budget: Llama 3's 3.17 to 3.94 characters per token [12] buys a quadratic saving for a linear cost in embedding parameters, whereas the 15-fold cross-language disparity [13] makes the effective context window a function of input language rather than architecture. An architectural choice invalidates a training rule: once routing decouples capacity from compute, Chinchilla's equal-scaling relationship no longer specifies a token budget, and DeepSeek-V3's 22 tokens per total parameter against 400 per activated parameter [36] shows the ambiguity is not marginal. An adaptation method's cost is paid in whichever currency the layer beneath is short of—LoRA in weights, adapters in latency, prefix tuning in the very context Section 2.1.3 shows to be scarce.

Engineering progress worsens the alignment problem rather than easing it. Hardware, system, and data constraints bound *how much* can be trained, and every advance surveyed relaxes them. The objective tier does not behave this way. Because reward hacking follows predictable scaling laws in which optimising a learned proxy too far degrades true performance—Goodhart's law in operation [70]—additional optimisation capability reaches that failure faster. The methods reducing alignment's cost—DPO [71], KTO [72], Constitutional AI [73]—each remove a different expense while each retains a learned proxy, and therefore, the failure mode. Alignment is the one layer where these advances are actively adverse: they make it cheaper to optimise hard against an objective known to be approximate.

Limitations. The configuration guidance of Section 6 derives from published figures and the constraints established here, not from our own benchmarking, and no controlled comparison across scales under a common protocol is known to us. The comparative tables draw on figures reported against differing base models and protocols, and are indicative of design points rather than commensurable—a limitation of the literature we mark rather than conceal. Finally, the FP8 treatment reflects hardware available at the time of writing, and the vendor recommendations [37] are precisely those whose satisfaction would date it.

The foundation surveyed here—subword tokenisation, scaled dot-product attention, normalisation placement, mixed-precision training, hybrid parallelism, memory offloading, sparse routing, and parameter-efficient adaptation—is not a list of independent techniques but a chain in which each element's viability is set by its neighbours. The consequence is that decisions which appear local are not: a vocabulary chosen for compression fixes an attention budget; a routing scheme adopted for compute efficiency invalidates the token budget

inherited from a scaling law; an adaptation method selected for its parameter count is charged against whichever resource the tier beneath it has least of. None of these couplings is visible from within the chapter that introduces either end of it, which is why a survey organised by component, however comprehensive, has nowhere to state them, and why we have organised by the dependency instead. That choice is also the part of this account we expect to age best: individual methods will be superseded, whereas the structure of the dependencies between them is set by the arithmetic and the memory hierarchy, and changes far more slowly. The contribution we claim is the chain, not the elements.

Author Contributions: Conceptualisation, M.L. and F.D.; investigation (literature review and analysis), M.L., J.I., M.D., E.Z., M.K., I.T., T.J. and R.P.; writing—original draft preparation, M.L.; writing—review and editing, M.L., F.D., J.I., M.D., E.Z., M.K., I.T., T.J. and R.P.; visualisation, M.L., T.J. and R.P.; supervision, F.D.; project administration, F.D.; funding acquisition, F.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I05-03-V02-00039.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: During the preparation of this manuscript, the authors used Claude (Anthropic, <https://claude.ai>) for the purposes of language editing and grammar correction throughout the manuscript. The authors have reviewed and edited the output and take full responsibility for the content of this publication. All scientific content, analysis, and conclusions are the authors' own.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BF16	Brain Floating Point (16-bit)
BN	Batch Normalisation
BPE	Byte-Pair Encoding
CAI	Constitutional AI
DNN	Deep Neural Network
DP	Data Parallelism
DPO	Direct Preference Optimisation
DPR	Dense Passage Retrieval
FP8	8-bit Floating Point
FP16	16-bit Floating Point
FP32	32-bit Floating Point
GEMM	General Matrix Multiplication
GPU	Graphics Processing Unit
HALO	Human-Aware Loss
HBM	High-Bandwidth Memory
KTO	Kahneman–Tversky Optimisation
LLM	Large Language Model
LN	Layer Normalisation
LoRA	Low-Rank Adaptation
MoE	Mixture-of-Experts
NF4	4-bit NormalFloat
OOV	Out-of-Vocabulary

PEFT	Parameter-Efficient Fine-Tuning
PMP	Pipeline Model Parallelism
PPO	Proximal Policy Optimisation
QLoRA	Quantised Low-Rank Adaptation
RAG	Retrieval-Augmented Generation
RLAIF	Reinforcement Learning from AI Feedback
RLHF	Reinforcement Learning from Human Feedback
RMSNorm	Root Mean Square Layer Normalisation
RoPE	Rotary Position Embedding
SFT	Supervised Fine-Tuning
SRAM	Static Random-Access Memory
TMP	Tensor Model Parallelism

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, 4–9 December 2017; Volume 30.
2. Minaee, S.; Mikolov, T.; Nikzad, N.; Chenaghlu, M.; Socher, R.; Amatriain, X.; Gao, J. Large Language Models: A Survey. *arXiv* **2024**, arXiv:2402.06196.
3. Raiaan, M.A.K.; Mukta, M.S.H.; Fatema, K.; Fahad, N.M.; Sakib, S.; Mim, M.M.J.; Ahmad, J.; Ali, M.E.; Azam, S. A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges. *IEEE Access* **2024**, *12*, 26839–26874. <https://doi.org/10.1109/ACCESS.2024.3365742>.
4. Liu, Y.; He, H.; Han, T.; Zhang, X.; Liu, M.; Tian, J.; Zhang, Y.; Wang, J.; Gao, X.; Zhong, T.; et al. Understanding LLMs: A Comprehensive Overview from Training to Inference. *arXiv* **2024**, arXiv:2401.02038.
5. Kudo, T. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), Melbourne, Australia, 15–20 July 2018; pp. 66–75.
6. Kudo, T.; Richardson, J. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations, Brussels, Belgium, 31 October–4 November 2018; pp. 66–71. <https://doi.org/10.18653/v1/D18-2012>.
7. Sennrich, R.; Haddow, B.; Birch, A. Neural Machine Translation of Rare Words with Subword Units. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Berlin, Germany, 7–12 August 2016; pp. 1715–1725. <https://doi.org/10.18653/v1/P16-1162>.
8. Kozma, L.; Voderholzer, J. Theoretical Analysis of Byte-Pair Encoding. *arXiv* **2024**, arXiv:2411.08671.
9. Provilkov, I.; Emelianenko, D.; Voita, E. BPE-Dropout: Simple and Effective Subword Regularization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL), Virtual, 5–10 July 2020; pp. 1882–1892. <https://doi.org/10.18653/v1/2020.acl-main.170>.
10. Schuster, M.; Nakamura, K. Japanese and Korean Voice Search. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Kyoto, Japan, 25–30 March 2012; pp. 5149–5152. <https://doi.org/10.1109/ICASSP.2012.6289079>.
11. Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv* **2016**, arXiv:1609.08144.
12. Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. The Llama 3 Herd of Models. *arXiv* **2024**, arXiv:2407.21783.
13. Petrov, A.; La Malfa, E.; Torr, P.H.S.; Bibi, A. Language Model Tokenizers Introduce Unfairness Between Languages. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023; Volume 36.
14. Su, J.; Lu, Y.; Pan, S.; Murtadha, A.; Wen, B.; Liu, Y. RoFormer: Enhanced Transformer with Rotary Position Embedding. *Neurocomputing* **2024**, *568*, 127063. <https://doi.org/10.1016/j.neucom.2023.127063>.
15. Luong, M.-T.; Pham, H.; Manning, C.D. Effective Approaches to Attention-Based Neural Machine Translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), Lisbon, Portugal, 17–21 September 2015; pp. 1412–1421. <https://doi.org/10.18653/v1/D15-1166>.

16. Szandala, T. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. In *Bio-Inspired Neurocomputing*; Bhoi, A.K., Mallick, P.K., Liu, C.-M., Balas, V.E., Eds.; Studies in Computational Intelligence; Springer: Singapore, 2021; Volume 903. https://doi.org/10.1007/978-981-15-5495-7_11.
17. Hendrycks, D.; Gimpel, K. Gaussian Error Linear Units (GELUs). *arXiv* **2016**, arXiv:1606.08415.
18. Shazeer, N. GLU Variants Improve Transformer. *arXiv* **2020**, arXiv:2002.05202.
19. Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv* **2023**, arXiv:2302.13971.
20. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer Normalization. *arXiv* **2016**, arXiv:1607.06450.
21. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on Machine Learning (ICML), Lille, France, 7–9 July 2015; pp. 448–456.
22. Zhang, B.; Sennrich, R. Root Mean Square Layer Normalization. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 8–14 December 2019; Volume 32.
23. Xiong, R.; Yang, Y.; He, D.; Zheng, K.; Zheng, S.; Xing, C.; Zhang, H.; Lan, Y.; Wang, L.; Liu, T.-Y. On Layer Normalization in the Transformer Architecture. In Proceedings of the 37th International Conference on Machine Learning (ICML), Virtual, 13–18 July 2020; pp. 10524–10533.
24. Rajbhandari, S.; Rasley, J.; Ruwase, O.; He, Y. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Atlanta, GA, USA, 9–19 November 2020; pp. 1–16.
25. Li, M.; Andersen, D.G.; Park, J.W.; Smola, A.J.; Ahmed, A.; Josifovski, V.; Long, J.; Shekita, E.J.; Su, B.-Y. Scaling Distributed Machine Learning with the Parameter Server. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Broomfield, CO, USA, 6–8 October 2014; pp. 583–598.
26. Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; Catanzaro, B. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv* **2020**, arXiv:1909.08053.
27. Guan, L.; Li, D.-S.; Liang, J.-Y.; Wang, W.-J.; Ge, K.-S.; Lu, X.-C. Advances of Pipeline Model Parallelism for Deep Learning Training: An Overview. *J. Comput. Sci. Technol.* **2024**, *39*, 567–584. <https://doi.org/10.1007/s11390-024-3872-3>.
28. Harlap, A.; Narayanan, D.; Phanishayee, A.; Seshadri, V.; Devanur, N.; Ganger, G.; Gibbons, P. PipeDream: Fast and Efficient Pipeline Parallel DNN Training. *arXiv* **2018**, arXiv:1806.03377.
29. Yeung, M.T.; Qi, P.; Lin, M.; Wan, X. Balancing Pipeline Parallelism with Vocabulary Parallelism. *arXiv* **2024**, arXiv:2411.05288.
30. Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, M.X.; Chen, D.; Lee, H.; Ngiam, J.; Le, Q.V.; Wu, Y.; et al. GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism. *arXiv* **2018**, arXiv:1811.06965.
31. Micikevicius, P.; Narang, S.; Alben, J.; Diamos, G.; Elsen, E.; Garcia, D.; Ginsburg, B.; Houston, M.; Kuchaiev, O.; Venkatesh, G.; et al. Mixed Precision Training. In Proceedings of the International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018.
32. Le Scao, T.; Fan, A.; Akiki, C.; Pavlick, E.; Ilić, S.; Hesslow, D.; Castagné, R.; Luccioni, A.S.; Yvon, F.; Gallé, M.; et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *arXiv* **2023**, arXiv:2211.05100.
33. Rae, J.W.; Borgeaud, S.; Cai, T.; Millican, K.; Hoffmann, J.; Song, F.; Aslanides, J.; Henderson, S.; Ring, R.; Young, S.; et al. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. *arXiv* **2022**, arXiv:2112.11446.
34. Micikevicius, P.; Stosic, D.; Burgess, N.; Cornea, M.; Dubey, P.; Grisenthwaite, R.; Ha, S.; Heinecke, A.; Judd, P.; Kamalu, J.; et al. FP8 Formats for Deep Learning. *arXiv* **2022**, arXiv:2209.05433.
35. Dettmers, T.; Pagnoni, A.; Holtzman, A.; Zettlemoyer, L. QLoRA: Efficient Finetuning of Quantized LLMs. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023; Volume 36.
36. Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. DeepSeek-V3 Technical Report. *arXiv* **2024**, arXiv:2412.19437.
37. Zhao, C.; Deng, C.; Ruan, C.; Dai, D.; Gao, H.; Li, J.; Zhang, L.; Huang, P.; Zhou, S.; Ma, S.; et al. Insights into DeepSeek-V3: Scaling Challenges and Reflections on Hardware for AI Architectures. In Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA), Tokyo, Japan, 21–25 June 2025.
38. Krizhevsky, A. One Weird Trick for Parallelizing Convolutional Neural Networks. *arXiv* **2014**, arXiv:1404.5997.
39. Ben-Nun, T.; Hoefler, T. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *ACM Comput. Surv.* **2019**, *52*, 65. <https://doi.org/10.1145/3320060>.
40. Akintoye, S.B.; Han, L.; Zhang, X.; Chen, H.; Zhang, D. A Hybrid Parallelization Approach for Distributed and Scalable Deep Learning. *IEEE Access* **2022**, *10*, 77950–77961. <https://doi.org/10.1109/ACCESS.2022.3193690>.
41. Li, S.; Hoefler, T. Chimera: Efficiently Training Large-Scale Neural Networks with Bidirectional Pipelines. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), St. Louis, MO, USA, 14–19 November 2021; pp. 1–14.

42. Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Ranzato, M.A.; Senior, A.; Tucker, P.; Yang, K.; et al. Large Scale Distributed Deep Networks. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Lake Tahoe, NV, USA, 3–6 December 2012; Volume 25.
43. Ren, J.; Rajbhandari, S.; Aminabadi, R.Y.; Ruwase, O.; Yang, S.; Zhang, M.; Li, D.; He, Y. ZeRO-Offload: Democratizing Billion-Scale Model Training. In Proceedings of the 2021 USENIX Annual Technical Conference (ATC), Virtual, 14–16 July 2021; pp. 551–564.
44. Li, S.; Liu, H.; Bian, Z.; Fang, J.; Huang, H.; Liu, Y.; Wang, B.; You, Y. Colossal-AI: A Unified Deep Learning System for Large-Scale Parallel Training. In Proceedings of the 37th International Conference on Supercomputing (ICS), Orlando, FL, USA, 21–23 June 2023; pp. 84–94.
45. Lai, Z.; Li, S.; Tang, X.; Ge, K.; Liu, W. Merak: An Efficient Distributed DNN Training Framework With Automated 3D Parallelism for Giant Foundation Models. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 1466–1478. <https://doi.org/10.1109/TPDS.2023.3247001>.
46. Rajbhandari, S.; Ruwase, O.; Rasley, J.; Smith, S.; He, Y. ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), St. Louis, MO, USA, 14–19 November 2021. <https://doi.org/10.1145/3458817.3476205>.
47. Chen, S.; Wang, Z.; Guan, Z.; Liu, Y.; Gibbons, P.B. Practical Offloading for Fine-Tuning LLM on Commodity GPU via Learned Sparse Projectors. *arXiv* **2024**, arXiv:2406.10181.
48. Chen, T.; Xu, B.; Zhang, C.; Guestrin, C. Training Deep Nets with Sublinear Memory Cost. *arXiv* **2016**, arXiv:1604.06174.
49. Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling Laws for Neural Language Models. *arXiv* **2020**, arXiv:2001.08361.
50. Hoffmann, J.; Borgeaud, S.; Mensch, A.; Buchatskaya, E.; Cai, T.; Rutherford, E.; Casas, D.D.; Hendricks, L.A.; Welbl, J.; Clark, A.; et al. Training Compute-Optimal Large Language Models. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022; Volume 35.
51. Clark, A.; de las Casas, D.; Guy, A.; Mensch, A.; Paganini, M.; Hoffmann, J.; Damoc, B.; Hechtman, B.; Cai, T.; Borgeaud, S.; et al. Unified Scaling Laws for Routed Language Models. In Proceedings of the 39th International Conference on Machine Learning (ICML), Baltimore, MD, USA, 17–23 July 2022; pp. 4057–4086.
52. Muennighoff, N.; Rush, A.M.; Barak, B.; Le Scao, T.; Piktus, A.; Tazi, N.; Pyysalo, S.; Wolf, T.; Raffel, C. Scaling Data-Constrained Language Models. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023; Volume 36.
53. Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; Choi, Y. The Curious Case of Neural Text Degeneration. In Proceedings of the International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, 26–30 April 2020.
54. Li, X.L.; Holtzman, A.; Fried, D.; Liang, P.; Eisner, J.; Hashimoto, T.; Zettlemoyer, L.; Lewis, M. Contrastive Decoding: Open-ended Text Generation as Optimization. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL), Toronto, ON, Canada, 9–14 July 2023; pp. 12286–12312.
55. Leviathan, Y.; Kalman, M.; Matias, Y. Fast Inference from Transformers via Speculative Decoding. In Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July 2023; pp. 19274–19286.
56. Willard, B.T.; Louf, R. Efficient Guided Generation for Large Language Models. *arXiv* **2023**, arXiv:2307.09702.
57. Parthasarathy, V.B.; Zafar, A.; Khan, A.; Shahid, A. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. *arXiv* **2024**, arXiv:2408.13296.
58. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. In Proceedings of the International Conference on Learning Representations (ICLR), Virtual, 25–29 April 2022.
59. Li, X.L.; Liang, P. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL), Bangkok, Thailand, 1–6 August 2021; pp. 4582–4597. <https://doi.org/10.18653/v1/2021.acl-long.353>.
60. Lester, B.; Al-Rfou, R.; Constant, N. The Power of Scale for Parameter-Efficient Prompt Tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), Punta Cana, Dominican Republic, 7–11 November 2021; pp. 3045–3059. <https://doi.org/10.18653/v1/2021.emnlp-main.243>.
61. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-Efficient Transfer Learning for NLP. In Proceedings of the 36th International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; pp. 2790–2799.
62. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.T.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Virtual, 6–12 December 2020; Volume 33, pp. 9459–9474.
63. Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, M.; Wang, H. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv* **2023**, arXiv:2312.10997.

64. Sarthi, P.; Abdullah, S.; Tuli, A.; Khanna, S.; Goldie, A.; Manning, C.D. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
65. Shaukat, M.A.; Adnan, M.; Kuhn, C.C.N. A Systematic Investigation of Document Chunking Strategies and Embedding Sensitivity. *arXiv* **2026**, arXiv:2603.06976.
66. Khattab, O.; Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual, 25–30 July 2020; pp. 39–48. <https://doi.org/10.1145/3397271.3401075>.
67. Asai, A.; Wu, Z.; Wang, Y.; Sil, A.; Hajishirzi, H. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
68. Jiang, H.; Wu, Q.; Lin, C.-Y.; Yang, Y.; Qiu, L. LLMingua: Compressing Prompts for Accelerated Inference of Large Language Models. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP), Singapore, 6–10 December 2023.
69. Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. Training Language Models to Follow Instructions with Human Feedback. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022; Volume 35, pp. 27730–27744.
70. Gao, L.; Schulman, J.; Hilton, J. Scaling Laws for Reward Model Overoptimization. In Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July 2023; pp. 10835–10866.
71. Rafailov, R.; Sharma, A.; Mitchell, E.; Ermon, S.; Manning, C.D.; Finn, C. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 10–16 December 2023; Volume 36.
72. Ethayarajh, K.; Xu, W.; Muennighoff, N.; Jurafsky, D.; Kiela, D. KTO: Model Alignment as Prospect Theoretic Optimization. In Proceedings of the 41st International Conference on Machine Learning (ICML), Vienna, Austria, 21–27 July 2024.
73. Bai, Y.; Kadavath, S.; Kundu, S.; Askell, A.; Kernion, J.; Jones, A.; Chen, A.; Goldie, A.; Mirhoseini, A.; McKinnon, C.; et al. Constitutional AI: Harmlessness from AI Feedback. *arXiv* **2022**, arXiv:2212.08073.
74. Guo, D.; Yang, D.; Zhang, H.; Song, J.; Wang, P.; Zhu, Q.; Xu, R.; Zhang, R.; Ma, S.; Bi, X.; et al. DeepSeek-R1 Incentivizes Reasoning in LLMs through Reinforcement Learning. *Nature* **2025**, *645*, 633–638. <https://doi.org/10.1038/s41586-025-09422-z>.
75. Frantar, E.; Alistarh, D. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. In Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, HI, USA, 23–29 July 2023.
76. Sun, M.; Liu, Z.; Bair, A.; Kolter, J.Z. A Simple and Effective Pruning Approach for Large Language Models. In Proceedings of the International Conference on Learning Representations (ICLR), Vienna, Austria, 7–11 May 2024.
77. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning Both Weights and Connections for Efficient Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 7–12 December 2015; Volume 28.
78. Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; Dean, J. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In Proceedings of the International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017.
79. Jiang, A.Q.; Sablayrolles, A.; Roux, A.; Mensch, A.; Savary, B.; Bamford, C.; Chaplot, D.S.; Casas, D.D.; Hanna, E.B.; Bressand, F.; et al. Mixtral of Experts. *arXiv* **2024**, arXiv:2401.04088.
80. Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. Qwen3 Technical Report. *arXiv* **2025**, arXiv:2505.09388.
81. Dao, T.; Fu, D.Y.; Ermon, S.; Rudra, A.; Ré, C. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 28 November–9 December 2022; Volume 35, pp. 16344–16359.
82. Shazeer, N. Fast Transformer Decoding: One Write-Head is All You Need. *arXiv* **2019**, arXiv:1911.02150.
83. Ainslie, J.; Lee-Thorp, J.; de Jong, M.; Zemlyanskiy, Y.; Lebrón, F.; Sanghai, S. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP), Singapore, 6–10 December 2023.
84. Yuan, J.; Gao, H.; Dai, D.; Luo, J.; Zhao, L.; Zhang, Z.; Xie, Z.; Wei, Y.X.; Wang, L.; Xiao, Z.; et al. Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention. *arXiv* **2025**, arXiv:2502.11089.
85. Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C.H.; Gonzalez, J.E.; Zhang, H.; Stoica, I. Efficient Memory Management for Large Language Model Serving with PagedAttention. In Proceedings of the 29th Symposium on Operating Systems Principles (SOSP), Koblenz, Germany, 23–26 October 2023.
86. Frantar, E.; Ashkboos, S.; Hoeffler, T.; Alistarh, D. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. In Proceedings of the International Conference on Learning Representations (ICLR), Kigali, Rwanda, 1–5 May 2023.

87. Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; Han, S. AWQ: Activation-Aware Weight Quantization for LLM Compression and Acceleration. In Proceedings of the Machine Learning and Systems (MLSys), Santa Clara, CA, USA, 13–16 May 2024.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.