

Large Language Model Compression for Autoregressive Text Generation: Methods, Evaluation, and Deployment Trade-Offs

Chenhui Zhu ¹, Runhua Zhang ^{2,*} and Fangzheng Tian ¹

¹ School of Computer Science and Engineering, Beihang University, Beijing 100191, China; zhuch0401@buaa.edu.cn (C.Z.); tianfangzheng@buaa.edu.cn (F.T.)

² School of Artificial Intelligence, Beihang University, Beijing 100191, China

* Correspondence: rhzhang20@buaa.edu.cn

Abstract

Compression changes both the information retained by a large language model and the resources required to complete a request. This narrative review examines quantization, pruning and sparsity, knowledge distillation, low-rank factorization, and token, context, and key–value cache compression for autoregressive text generation, principally with decoder-only LLMs. Its central contribution is a synthesis of changes relative to each study's baseline, connecting compression mechanisms to capability and deployment choices. Four numerical tables distinguish quality changes at stated compression budgets, distillation gains over same-size students, and paired or independent deployment measurements. Reporting is appraised by source identity, comparison conditions, measurement scope, artefacts, and uncertainty. The reviewed cases illustrate how capacity constraints change the feasible precision choices, how allocating a shared budget changes different task scores, and how compact-model construction and request compression address complementary resource demands. A deployment-validation sequence turns these findings into candidate checks for a target workload. Remaining questions concern transfer across capabilities, delayed use of compressed information, interactions between compression stages, and complete-task cost, including agent interactions.

Keywords: large language models; model compression; quantization; pruning; knowledge distillation; context compression; efficient inference

1. Introduction

Large language model (LLM) compression must preserve useful behaviour while reducing the resources needed to deliver it. Autoregressive generation repeatedly accesses persistent weights and a key–value (KV) cache storing attention states from preceding tokens; cache demand grows with retained context and concurrent sequences [1,2]. Weight storage stays fixed for a loaded model, while the expanding cache can exhaust the remaining device memory and limit context length or concurrency. The resulting quality changes and execution benefits depend on which information is retained and how it is represented and used. The central question of this review is how to connect a compression choice to retained capability and resource savings under a stated deployment constraint.

Here, compression means reducing the resources used to represent or process a model and its requests by lowering precision, reducing stored or active structure, compacting retained information, or shortening generated trajectories. The main method families make different decisions. Quantization changes numerical representation, pruning and sparsity



Academic Editor: Arkaitz Zubiaga

Received: 2 October 2026

Revised: 15 October 2026

Accepted: 16 October 2026

Published: 19 October 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 \(CC BY\) license](https://creativecommons.org/licenses/by/4.0/).

select parameters or computation, and low-rank factorization restricts transformations to smaller subspaces. Distillation supplies supervision through which a compact student learns or recovers teacher behaviour [3]. Request-state and input compression extend these decisions to history and prompt content [4–6]. Learned context representations provide a further choice of information channel, while reasoning-output compression reduces the trajectory generated to answer a question [7,8]. The categories overlap because a deployed model can use a distilled architecture, quantized weights, and a compressed cache together. Interpreting such combinations requires separating the object changed from the mechanism used to change it.

Existing reviews provide substantial coverage of this landscape. Zhu et al. survey the four model-compression families together with metrics and benchmarks [3]. Wan et al. review efficient LLMs from model, data, and framework perspectives, covering compression alongside training, adaptation, and serving [9]. Zhou et al. organize inference optimization at data, model, and system levels and include comparative experiments on representative techniques [10]. Li et al. organize KV management at token, model, and system levels [11]. Chen et al. examine quantization across the model lifecycle and its kernel implementation, while the on-device survey by Chen et al. connects compression to system optimization and accuracy–latency–energy–memory evaluation [12,13]. Li et al. further organize prompt compression into hard and soft approaches [14]. Table 1 selects surveys spanning broad model compression, inference efficiency, numerical representation, request state, and prompt compression. This coverage provides complementary reference points for the review’s main contribution: a synthesis that links compression mechanisms to source-relative quality costs, measured resources, and deployment choices. The organizing fields are the object changed, the capability evaluated, and the resource measured.

Table 1. Positioning relative to selected surveys, summarized by their organizing emphasis.

Review	Organizing Emphasis	Role in This Review’s Positioning
Zhu et al. (2024) [3]	Compression families, metrics, and benchmarks	Broad methodological foundation
Wan et al. (2024) [9]	Model-, data-, and framework-centric efficiency	Broad efficiency landscape spanning training and inference
Zhou et al. (2024) [10]	Data-, model-, and system-level inference optimization; comparative experiments	Inference bottlenecks and interactions across optimization levels
Li et al. (2025) [11]	Token-, model-, and system-level KV management	Detailed request-state perspective
Chen et al. (2026), quantization [12]	Quantization lifecycle and hardware execution	Numerical representation and kernel perspective
Chen et al. (2026), on-device [13]	Compression, systems, and multi-resource evaluation	Deployment and evaluation perspective
Li et al. (2025), prompt compression [14]	Hard and soft prompt compression	Input and learned-representation perspective
This review	Mechanisms, source-relative quality evidence, and execution scope	Budget allocation, student recovery, and deployment checks

Two interpretation problems motivate that sequence. First, equal nominal compression levels can conceal different approximations. A sparse pattern constrains which values can survive; a composite representation divides storage among a backbone and corrections. Second, a resource result can describe a different setting from its associated quality test.

Palu’s cache representations and QServe’s format–system co-design demonstrate why state size, reconstruction, and implementation must be considered together [5,15]. Capability evidence has its own scope: The Pitfalls of KV Cache Compression finds unequal effects across instruction categories, and ACBench tests compressed models on task families including interactive behaviour [16,17]. Context-compression evaluations likewise distinguish task scores from grounding, entity retention, and precise recall [18,19]. These findings motivate a capability profile matched to the intended use.

The synthesis proceeds from a method map, through reported trade-offs, to candidate checks for a target workload:

1. A map of compression targets and interventions. The compression-target map locates interventions on model parameters, structure, activations, input representations, KV state, and generated reasoning within preparation and inference. Teacher supervision appears separately as a way to construct or recover a model. Sections 3–7 explain the corresponding approximation choices and execution requirements.
2. Source-relative quantitative evidence. The quantitative synthesis relates compression budgets to quality changes and distinguishes paired from independent deployment measurements. Distillation comparisons evaluate teacher-guided improvements over same-size students, with model-size savings referenced to the teacher. Together, they separate representation loss, student recovery, and execution benefit.
3. Constraint-based candidate checks. The deployment checklist maps eight application constraints to interventions, access requirements, and minimum measurements. Section 8.3 applies these checks to a reported hardware and workload setting; Section 8.4 draws out their implications for capacity screening, capability-based allocation, and combined model/request compression.

Figure 1 outlines the review’s progression from compression mechanisms to deployment choices. Section 2 establishes the scope, taxonomy, and comparison criteria for the reviewed literature. Sections 3–6 examine quantization, pruning and sparsity, distillation, and low-rank factorization; Section 7 extends the analysis to inputs, context, the KV state, and generated reasoning. Section 8 combines this evidence to assess method combinations under quality, capacity, and latency constraints. Sections 9 and 10 present the open challenges and principal conclusions, respectively.

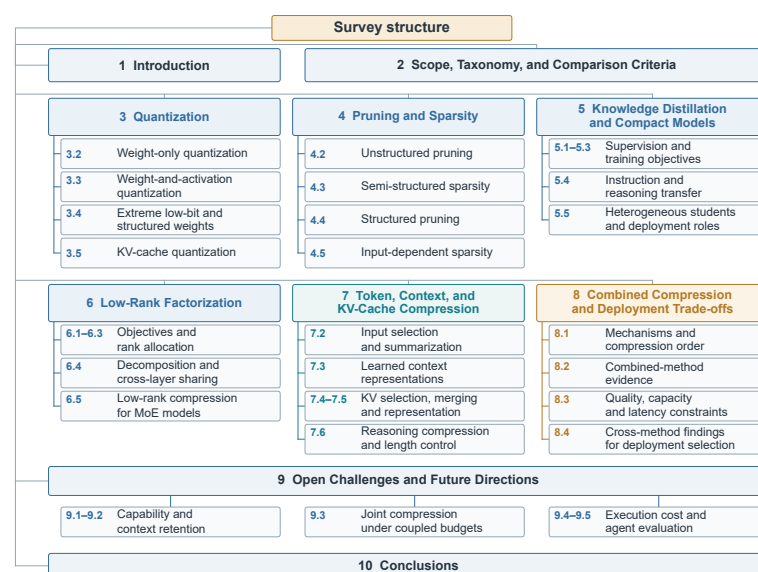


Figure 1. Organization of the review with selected subsection themes. Number ranges group related subsections.

2. Scope, Taxonomy, and Comparison Criteria

2.1. Scope of the Review

This narrative review examines compression for autoregressive text generation, principally with decoder-only LLMs. The core families are quantization, pruning and sparsity, knowledge distillation, and low-rank factorization. The scope extends to input text, learned context representations, KV state, and generated reasoning length because they change the resources required to complete a request. Earlier encoder and encoder–decoder studies provide methodological origins, and MoE examples clarify stored versus active capacity. Recovery training and assisted decoding are discussed as roles within compression pipelines; agent tasks appear as a future application in Section 9.5. General training acceleration, parameter-efficient adaptation, multimodal architectures, and unrestricted test-time-compute optimization fall outside the main taxonomy.

2.2. Literature Search, Study Selection, and Bias Control

The literature search covered publications from 2023–2026 in Scopus, Web of Science, and arXiv, supplemented by citation searching and reviewer recommendations. Selected earlier studies provided methodological background. Retained records document thematic searches on 28 and 30 September 2026 and a targeted update on 12 October 2026. Examples include “KV cache compression eviction merging LLM”, “prompt compression information preservation”, and “large language model quantization rotation 2025 2026”. These examples are topic queries used in the search and update process. Publisher pages, proceedings, and author repositories supplied source identification and technical evidence.

Eligibility required an identifiable source explaining a compression mechanism, retained capability, resource use, or a relevant deployment condition. Original studies supplied technical evidence; surveys and foundational works supported positioning and methodological context. Duplicate versions and studies outside these criteria were excluded. Two authors screened the literature, with disagreements resolved through discussion and adjudication with the third author. Titles, DOIs, and arXiv identifiers linked versions of a study across chapters. Both published papers and preprints were eligible. Formal publication records established bibliographic identity where available, while the version inspected supplied the numerical evidence; material version differences are identified with the corresponding results.

Figure 2 summarizes the initial screening round. The searches identified 689 records: 247 from Scopus, 74 from Web of Science, and 368 from arXiv. Removing 77 duplicates left 612 records for abstract screening. After 448 exclusions, 164 reports underwent full-text assessment, of which 82 were excluded using the scope and relevance criteria above. Citation searching added 20 reports and reviewer recommendations added 3; all 23 were retrieved and assessed. The two routes yielded 105 included publications. The bibliography also contains supporting references and papers added in the numerical update described below.

Selection bias was addressed through complementary discovery routes, shared scope and relevance criteria across method families, and reconciliation of multiple versions of the same work. To limit selective interpretation, the synthesis considers reported quality losses, failure conditions, and preparation costs alongside resource savings.

The numerical tables were expanded retrospectively during revision, with source checks through 13 October 2026. The update selected cases from the covered method families with identifiable models, configurations, metrics, and source baselines, and it then inspected original method papers for deployment observations. Quantization and pruning quality cases prioritize Llama 2-7B to reduce architectural variation; student and request-compression blocks retain their source models because teacher–student pairs and request interfaces define the experiment. Severe quality losses and missing deployment

results remain visible. Cases lacking a usable baseline or a clearly defined resource were omitted from the corresponding relative comparison. This selection illustrates mechanisms and trade-offs across practical budgets.

The targeted literature-search cut-off is 12 October 2026; the numerical source registry was consolidated on 14 October 2026. Supplementary Evidence S1 identifies the publication record and the extraction version separately, with the source URL, result location, and checksum of the inspected PDF. Publisher-issued online-first articles count as published. A formal version is preferred when it contains the selected result; a retained arXiv version is identified explicitly when it supplies that result. Later revisions require a new record and recalculation of the affected comparison rather than replacement of isolated values. The registry covers the current numerical synthesis.

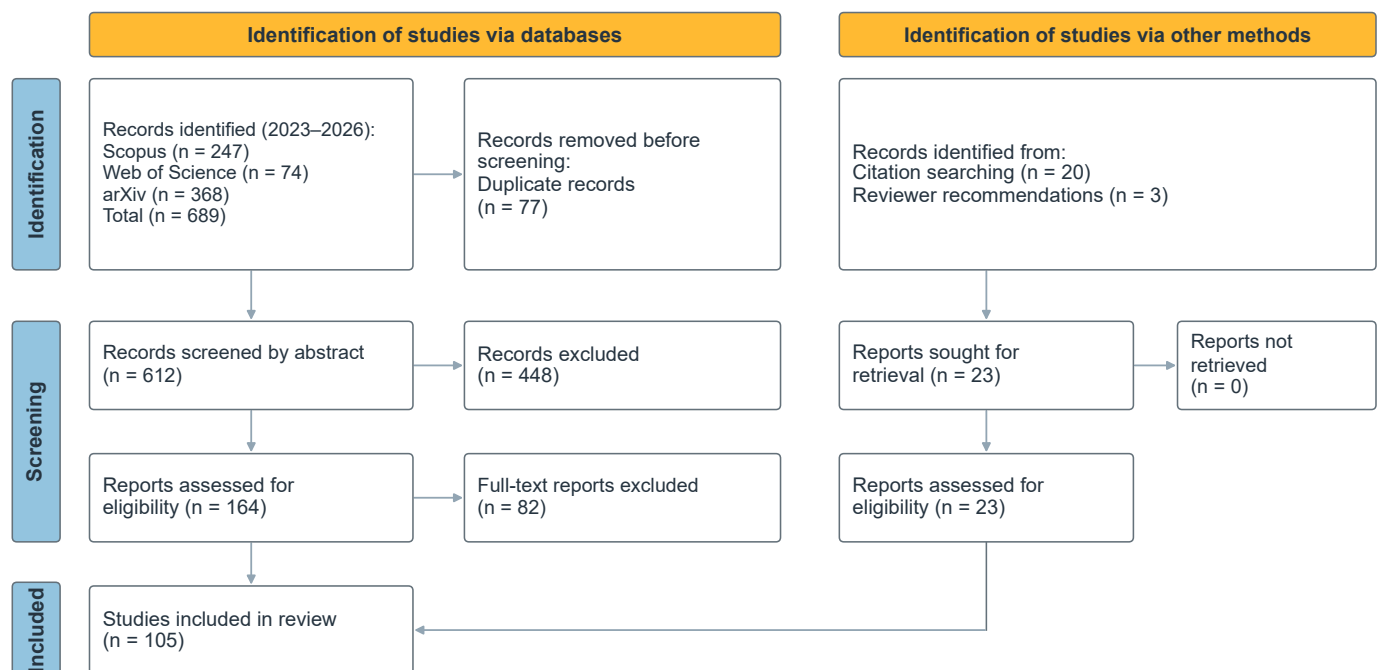


Figure 2. Literature identification and selection for this narrative review. The two routes distinguish database/repository searches from citation searching and reviewer recommendations. Counts refer to records before screening and to retained publications after version reconciliation. The layout adapts PRISMA 2020 flow-diagram conventions [20] (CC BY 4.0).

2.3. Compression Objects and Method Relationships

Compression objects and mechanisms are separate classification axes. Weights and KV states can each be quantized or factorized; pruning removes parameters or structures, while dynamic sparsity reduces the active subset. Distillation supplies supervision for constructing or recovering a compact model. Figure 3 separates persistent model parameters and structure from request-dependent information and generated output, with teacher supervision above the inference flow. Their compression effects are expressed through reduced representation, stored or active structure, retained history, or generated trajectory.

Persistent parameters and request-dependent state create different resource demands. Section 3.5 covers numerical KV representation; Section 7 covers retained positions, features, input evidence, and output length. These interventions require access ranging from a text interface to model weights, caches, or attention kernels. Execution optimizations provide a complementary layer: PagedAttention changes KV allocation and sharing, FlashAttention-2 changes attention work partitioning, and batching or offloading changes scheduling or placement [1,21–23]. They can improve execution without discarding represented

information. We discuss them where they explain whether a compressed representation produces a measured deployment benefit.

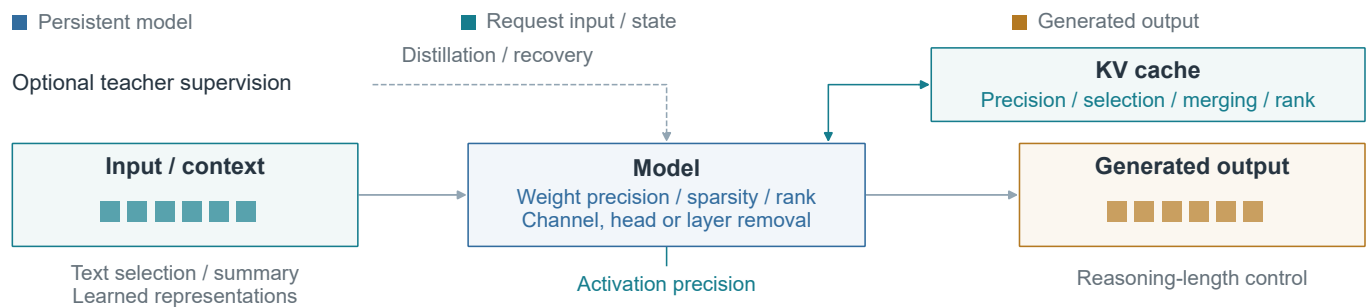


Figure 3. Compression targets in autoregressive generation. Blue denotes persistent model parameters and structure, teal denotes request-dependent inputs and states, and amber denotes generated output. Optional teacher supervision supports model construction or recovery. Solid arrows show inference data or state flow; dashed arrows show preparation. Labels identify interventions at each target.

2.4. Quality, Resources, and Comparable Evidence

Quality measures probe different properties. Reconstruction error measures a local approximation, perplexity (PPL) evaluates token prediction, and downstream scores test selected capabilities. Context evaluations additionally examine grounding, entity preservation, and precise recall [18,19]; output-compression studies pair answer correctness with actual generated length. Comparisons retain the model, data, metric, and source baseline. A configuration comprises its checkpoint, precision or structural budget, calibration/recovery procedure, and evaluation protocol. Source-relative changes express each result against its study's baseline, with task and runtime fields specifying its relevance to an application. Comparisons across sources retain differences in calibration, recovery, and evaluation protocols.

Table 2 summarizes the resource terminology used in the synthesis. Resource accounting distinguishes parameter count, stored weight bytes, KV payload, and peak process memory. Parameter count excludes precision; stored bytes include the representation and its metadata; process memory also includes buffers and runtime allocation. For any stated resource, the retained fraction is $\rho = R_c/R_b$, the removed fraction is $1 - \rho$, and the compression factor is $1/\rho$. Thus, a $4\times$ factor means 25% retained and 75% removed. Preparation includes calibration, decomposition, teacher generation, and recovery. Online selection, reconstruction, and budget estimation belong to request cost; offline document encoding and index storage have separate time and storage budgets.

Time to first token, time per output token, complete-request latency, and aggregate throughput use different timing boundaries. Input length determines prompt work and initial state; output length determines decoding steps and subsequent cache growth; batch size changes weight reuse and concurrent state occupancy. Deployment results retain these variables together with hardware, runtime, and baseline. Section 8.2 applies the distinctions to hybrid cases, and Section 8.3 connects them to candidate checks.

The quantitative synthesis uses the following transformations. For a resource amount R , the reduction is $100(1 - R_c/R_b)$; for latency T , speedup is T_b/T_c ; for throughput V , speedup is V_c/V_b . Relative perplexity change is $100(\text{PPL}_c/\text{PPL}_b - 1)$, and task change is $A_c - A_b$. Subscripts b and c denote the stated reference and evaluated configuration. For distillation, parameter savings use the teacher as reference, whereas quality changes use the same-size supervised fine-tuned (SFT) student. Accuracy and win-rate differences

use percentage points; ROUGE-L and LongBench differences use score points. These transformations retain the original metric and baseline within each experiment.

Table 2. Resource terminology used throughout the synthesis. Subscripts *b* and *c* denote baseline and compressed configurations.

Term	Definition and Accounting Boundary
Retained/removed fraction	$\rho = R_c/R_b/1 - \rho$, for the named resource.
Compression factor	$R_b/R_c = 1/\rho$; $4\times$ means 25% retained.
Nominal bit width/effective bits	Quantizer code width/ $8B_{\text{stored}}/N_{\text{original}}$; the latter includes reported metadata and exceptions.
Parameter count/weight bytes	Number of stored parameters/bytes for their representation, including metadata.
KV payload/peak memory	Retained key–value representation/maximum process allocation, including weights, cache, and runtime buffers.
Target/actual retention	Requested budget/observed retained positions or generated tokens relative to the stated reference.
Nominal/ideal/measured	Configured budget/representation calculation with stated exclusions/observed quantity under a specified workload.

Method citations identify originating papers; table notes identify the studies supplying quality comparisons. Positive perplexity changes and negative task-score changes indicate losses. S denotes a separate deployment experiment and P denotes paired quality and execution measurements. Resource entries distinguish measured savings, nominal budgets, ideal payloads, and analytical estimates. Ideal weight/KV payloads exclude metadata and higher-precision exceptions; weight-format estimates use reported effective bits. Kernel, block, model, and serving results retain their execution labels, including simulations.

2.5. Evidence Quality and Reproducibility Appraisal

The numerical synthesis is appraised along five reporting dimensions: (i) source identity, version and result location; (ii) the model, configuration, preparation, and comparison baseline; (iii) the resource boundary, hardware, runtime, and workload; (iv) reported code, checkpoints, or configuration artefacts; and (v) repetitions, seeds, and uncertainty. Fields are treated as reported, partly reported or unreported according to the information needed to interpret the selected result. These judgments remain dimension-specific: a code link documents an artefact route, while baseline and measurement details determine the scope of a comparison.

Supplementary Evidence S1 records the 63 quality configurations in the four numerical tables and their associated deployment cases. It separates method-origin citations from result-supplying studies, preserves baseline and compressed values, and records source-specific protocols and reporting gaps. The source appraisal and the kernel, block, model, or serving boundary provide an operational description of evidence coverage. A reported code link establishes an access route; replication on the intended workload remains a separate validation step.

Applying these checks changes the interpretation of several cases. DistiLLM reports means and standard deviations over five evaluation seeds for the OpenLLaMA2 comparison; the distillation synthesis uses changes in those means [24]. CALDERA’s selected rank–precision comparison supplies point estimates without uncertainty for the observed task reversal [25]. GPTQ’s OPT-175B decoding comparison changes the GPU count from five to one, so the gain includes placement effects [26]. OptiShear describes its CUTLASS-based latency evaluation as simulation, which remains explicit in the pruning synthesis [27].

QServe maximizes batch size separately by configuration, making its result a throughput-capacity comparison [15].

Artifact reporting also varies. QuIP# supplies code and prequantized-model links, while the inspected Tyr-the-Pruner version describes a future release [28,29]. Missing workload fields remain NR, quality-only cases retain an empty deployment entry, and separate experiments retain S. Small differences without uncertainty are interpreted descriptively. These checks guide the synthesis and the deployment-validation sequence in Section 8.3.

Uncertainty can arise from calibration sampling, generation seeds, prompt templates, benchmark samples, and evaluation-harness versions. Runtime also depends on warm-up, clocks and thermal state, CUDA and kernel versions, memory allocation, batching, and scheduling. Missing records for these factors limit claims about small gains and transfer between systems. Available repetitions and dispersion are preserved in Supplementary Evidence S1; absent variance is not reconstructed from rounded means.

3. Quantization

Quantization allocates a bit budget through error compensation, coordinate transformations, and richer representations for sensitive values. These choices affect retained quality and the operations needed to execute the compressed model. The following sections connect them to weights, activations, and cached attention states.

3.1. Scope and Design Dimensions

The compressed object determines the immediate resource effect. Weight-only quantization changes parameter storage while keeping activations at higher precision; weight-and-activation quantization also changes operands used in selected arithmetic operations; and KV quantization changes the persistent keys and values used during generation [2,26,30]. Labels such as W4A16 and W4A8KV4 identify nominal tensor precisions. Accumulators, exceptional values, and operators outside the quantized path can retain different precisions.

Figure 4 contrasts precision reduction with changes to connectivity and subspace dimension. Quantization changes the values representable in each entry, while preserving the tensor shape in this schematic.

A uniform scalar representation rounds a value onto a scaled grid as in Equation (1):

$$q(x) = \text{clip}\left(\text{round}(x/s) + z, 0, 2^b - 1\right), \quad \hat{x} = s(q(x) - z), \quad (1)$$

Here, $s > 0$ is a quantizer scale, z an integer zero point, and b the code width. The notation is generic: scales and zero points may be assigned per tensor, channel, or group, depending on the quantizer. Scale selection depends on calibration and the quantizer objective. Smaller groups adapt the grid to local distributions at the cost of metadata. Non-uniform scalar quantizers learn reconstruction values, while vector codebooks encode groups jointly [31,32]. For N weights stored in B_W bytes, $b_{\text{eff}} = 8B_W/N$ counts codes, scales, codebooks, sparse side information, and packing/alignment padding. Source-reported average rates retain their stated accounting convention. Activations, caches, and workspaces contribute separately to inference memory.

Preparation changes different variables across methods. Post-training quantization (PTQ) may optimize clipping or transformation parameters, as OmniQuant does through block reconstruction [33]. Quantization-aware training (QAT) adapts parameters under simulated quantization: LLM-QAT uses teacher-generated examples and distillation, with data-free meaning independence from the original training corpus [34]. EfficientQAT first optimizes all parameters within blocks and then tunes quantization step sizes end to end with the other parameters fixed [35].

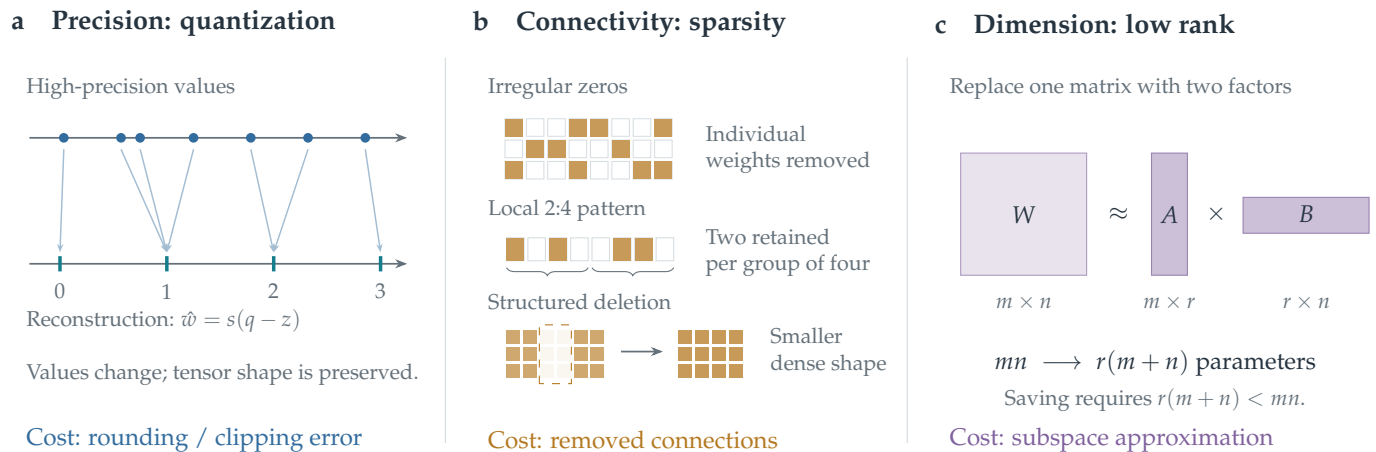


Figure 4. Three mathematical interventions and their approximation costs. (a) Quantization maps high-precision values to discrete codes. The unsigned codes 0–3 illustrate $b = 2$, consistent with Equation (1); s and z denote a scale and zero point in reconstruction. Tensor shape is preserved. (b) Individual zero entries, local 2:4 groups, and channel deletion impose different structural constraints. Filled cells are retained; the dashed region marks removed channels. (c) A rank- r factorization replaces $W \in \mathbb{R}^{m \times n}$ with AB . The parameter-count reduction shown excludes any correction residual or metadata. Values, dimensions, and patterns are illustrative.

3.2. Weight-Only Quantization

Weight-only quantization targets storage and weight traffic, particularly in batch-one decoding [26,31]. Three approaches to retaining quality are reconstruction, activation-aware scaling, and sensitivity-weighted representation.

GPTQ (OPTQ in the final ICLR paper) minimizes local output error on calibration inputs. Second-order reconstruction information compensates unquantized weights after each rounding step, with shared ordering, block updates, and a Cholesky reformulation enabling LLM-scale processing [26]. AWQ instead uses activation statistics to select channel scales, with compensating weight/input transformations preserving the linear map before rounding [36]. Its final representation protects salient channels through scaling without a separate FP16 branch or gradient-based weight reconstruction.

SqueezeLLM uses diagonal Fisher information to weight non-uniform clustering and optionally retains outliers and sensitive weights in an FP16 sparse component [31]. The additional representation capacity costs stored values, locations, and lookup tables. These sparse entries preserve selected weights, giving sparsity a different role from removal during pruning.

SqueezeLLM compares representations at the same reported average bit rate. For LLaMA-7B on C4, FP16 perplexity is 7.08. At 3.24 average bits, SqueezeLLM with a 0.45% sparse component gives 7.56, GPTQ 7.89, and AWQ 7.90 (Table 1 in [31]). GPTQ and AWQ use groups of 128 weights, and GPTQ includes activation ordering. The source evaluates them with its common perplexity protocol, using official AWQ checkpoints or code; calibration remains method-specific. SqueezeLLM estimates sensitivity from 100 samples. These conditions support a comparison of complete representations at a reported rate, with lower perplexity for the dense-and-sparse design in this experiment.

Execution depends on the operations required by each representation. AWQ supplies TinyChat, while SqueezeLLM supplies lookup and sparse kernels; QServe examines the cost of dequantization when serving at larger batch sizes [15,31,36]. A representation that preserves quality efficiently in bits still needs an execution path suited to its codes and exceptional values.

3.3. Weight-And-Activation Quantization

Large activation values stretch a uniform quantizer's range, leaving fewer useful reconstruction levels for typical values. SmoothQuant transfers part of this difficulty into weights through channel scaling. Using column-wise input samples X and $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ throughout, an invertible input-channel diagonal matrix S gives $WX = (WS)(S^{-1}X)$. This preserves the unquantized linear map while changing both operand distributions. SmoothQuant selects this balance to enable W8A8 computation [30]. The mechanism is a redistribution of range, so the scale choice determines how much quantization difficulty each operand carries.

OmniQuant jointly optimizes clipping and equivalent transformations through block reconstruction for weight-only and weight–activation quantization [33]. The optimized range and coordinate system determine the distributions seen by the quantizers.

Rotations distribute extreme coordinates across a transformed space. QuaRot combines computational invariances with Hadamard transformations; SpinQuant learns selected rotations while retaining online Hadamard operations [37,38]. FlatQuant uses learned affine transformations parametrized through Kronecker products and implements them with kernel fusion [39]. The practical distinction is where a transformation can be absorbed into parameters and where it must operate on incoming activations. Greater freedom in the transformed representation can therefore bring preparation work or online operations, even when the main matrix operands share the same nominal precision.

Preserving exceptions provides another response to outliers. QUIK routes most channels through low-bit computation and selected outliers through higher precision, with dedicated kernels [40]. Atom combines mixed precision with channel reordering, group-wise representation, and dynamic activation quantization for serving [41]. These designs retain a heterogeneous numerical path where distributing outliers across coordinates would use a different path.

3.4. Extreme Low-Bit and Structured Weight Representations

At very low rates, AQLM represents weight groups as sums of learned codebook vectors. Discrete assignments, codebook optimization, and block adjustment fit the representation [32]. QuIP# combines randomized Hadamard transforms, block-wise compensation, and lattice codes; QTIP uses trellis coding to link reconstruction choices across a code sequence [28,42]. These structured representations trade richer reconstruction choices against codebook or decoding costs.

3.5. KV-Cache Quantization

Cache quantization addresses a state that expands with requests and retained context. For a homogeneous decoder with unpadded sequences of the same retained length, the approximate payload is given by Equation (2):

$$M_{\text{KV}} \simeq B L T H_{\text{KV}} d_h \frac{b_K + b_V}{8} \text{ bytes}, \quad (2)$$

Here, B counts sequences, L layers, T retained tokens per sequence, H_{KV} heads shared by keys and values, and d_h dimensions per head; b_K, b_V are bits per key/value element. For unequal lengths, replace BT by $\sum_{i=1}^B T_i$. The equation counts one copy of the unpadded payload; metadata, protected high-precision regions, tensor-parallel replication, alignment padding, and allocator fragmentation add storage. For example, $B = 1, L = 32, T = 4096, H_{\text{KV}} = 32, d_h = 128$, and $b_K = b_V = 16$ give 2^{31} bytes (2 GiB). Quantizing both tensors to 4 bits gives 0.5 GiB of ideal payload at the same dimensions.

Keys and values need not share a quantization layout. KIVI uses per-channel keys and per-token values together with a higher-precision residual region [2]. KVQuant applies per-channel key quantization before rotary positional encoding and combines calibrated non-uniform representation with sparse outlier treatment [43]. Both exploit differences between the stored tensors, but they allocate precision through different representations and exception handling.

The streaming update policy adds another design axis. SKVQ combines channel reordering and clipped dynamic quantization with a recent window retained at higher precision [44]. MiniKV couples low-bit representation to persistent context selection and a compatible attention implementation [45]. Quantization alters the values of retained states; selection removes states from future attention. Their combination changes both representation fidelity and accessible history. Section 7 develops this latter distinction alongside merging, low-rank caches, and input compression.

3.6. From Compressed Formats to Executable Inference

Two execution problems recur: converting stored codes into arithmetic operands and organizing work around the target processor. QServe coordinates W4A8KV4 with progressive quantization and data layouts that reduce dequantization overhead in serving [15]. FLUTE targets lookup-table-quantized weights by restructuring them offline, vectorizing table access, and fusing dequantization into GPU matrix multiplication [46]. Both make representation decoding part of the matrix-operation design rather than an independent preprocessing step.

Other systems exploit the mathematical structure of their codes. LUT-GEMM uses binary-coded weights and lookup of reusable partial products with higher-precision activations [47]. T-MAC adapts bitwise decomposition and lookup layouts to CPU memory and register resources [48]. Quant-LLM's FP6-LLM design uses specialized packing and conversion for low-bit floating-point computation on GPUs [49].

Workload shape determines which path is attractive. Larger batches reuse weights more extensively, whereas additional sequences and longer contexts enlarge the KV state. Prefill and decoding also expose different matrix shapes. LLMC provides facilities for organizing compression configurations and evaluation [50].

3.7. Quantization Trade-Offs and Open Issues

Table 3 groups 20 quantization configurations into weight-only, weight–activation and joint weight–activation–KV cases. It reports source-relative changes in WikiText-2 perplexity and zero-shot accuracy, estimated weight-storage savings, and independent deployment results. Task codes link each quality comparison to its source; memory entries report running or peak memory, with block memory explicitly labelled.

In the AMQ-source weight-only group, GPTQ, AWQ, and AMQ imply approximately 75% weight-storage reduction, while their relative perplexity increases range from 3.8% to 11.3%. In the DuQuant-source W4A4 group, DuQuant and SmoothQuant show increases of 14.8% and 1419.6%, respectively. SmoothQuant's original W8A8 decoding result concerns a different precision. The W4A4 comparison shows that large quality differences remain between methods with the same nominal bit budget.

The deployment reports show where implementation benefits have been measured: GPTQ at whole-model decoding, AQLM at matrix–vector multiplication, and QuaRot and OSTQuant at transformer blocks. They guide selection of an execution path for a quality-qualified representation. Extending activation or KV quantization beyond weight-only storage also changes arithmetic or state access, motivating the distinct configuration groups in Table 3.

Table 3. Quantization: weight-storage savings and quality changes for Llama 2-7B relative to each source’s FP16 baseline, alongside independent deployment results.

Quality/Storage Case				Independent Deployment (S)
Method/Configuration	Weight Storage Saving	Δ PPL (%)	Δ task (pp)	Result/Tested Configuration
Weight-only				
GPTQ [26] W4; group NR	75.0% (ideal)	+11.3	−1.94 ^{A6}	3.24× decode OPT-175B; W3; batch 1; out 128 FP16: 5 A100s; W3: 1 A100
AWQ [36] W4; group NR	75.0% (ideal)	+6.6	−1.39 ^{A6}	3.73× generation TinyChat vs. HF FP16; RTX4090 Batch 1; in/out 4/200
AMQ [51] avg. W4.0 g128	75.0% (reported bits)	+3.8	−1.20 ^{A6}	Up to 2.67× decode L2-7B; bit sweep; L40S
SqueezeLLM [31] W4 A16 + 0.45% side	73.3% (format)	+1.8	–	Peak memory ↓ 68.5%; 1.70× speed L1-7B; A6000; batch 1; out 128
AQLM [32] W~4 A16	74.8% (format)	+1.8	−0.23 ^{Q4c}	1.57× matvec L2-7B; 2-bit; RTX3090
QuIP# [28] W4 A16	75.0% (ideal)	+1.4	+0.13 ^{Q4c}	3.21× generation L2-7B; W2; RTX4090
Weights and activations				
SmoothQuant [30] W4 A4 KV –	75.0% (ideal)	+1419.6	−19.20 ^{D6}	1.34× decode; mem. ↓ 47.4% OPT-30B; W8A8; A100 Batch 1; length 512
OS+ [52] W4 A4 KV –	75.0% (ideal)	–	−16.57 ^{D5}	1.51× generation OPT-13B; W8A8; hardware NR
OmniQuant [33] W4 A4 KV –	75.0% (ideal)	+160.7	−12.34 ^{D6}	1.94× decode; mem. ↓ 60.4% L1-7B; W4A16; A100
AffineQuant [53] W4 A4 KV –	75.0% (ideal)	+132.0	−11.08 ^{D6}	–
QLLM [54] W4 A4 KV –	75.0% (ideal)	+114.8	−10.63 ^{D5}	1.96× sequence throughput L1-7B; W4A4; RTX3090; length 2048
DuQuant [55] W4 A4 KV –	75.0% (ideal)	+14.8	−3.15 ^{D6}	1.93× prefill; mem. ↓ 68.7% L2-7B; batch 1; RTX3090

Table 3. Cont.

Quality/Storage Case				Independent Deployment (S)
Method/Configuration	Weight Storage Saving	Δ PPL (%)	Δ task (pp)	Result/Tested Configuration
Weights, activations and KV cache				
QuaRot [37]	75.0% (ideal)	+11.5	−4.78 ^{F6}	2.16× block prefill
W4 A4 KV4; GPTQ				L2-7B; RTX3090; batch 64; length 2048
SpinQuant [38]	75.0% (ideal)	+9.0	−3.56 ^{F6}	2.77× decode
W4 A4 KV4; GPTQ				L3-8B; W4A8; M1 Pro CPU
OSTQuant [56]	75.0% (ideal)	+8.2	−3.21 ^{F6}	2.14× block; block mem. ↓ 65.0%
W4 A4 KV4; GPTQ				L2-7B; RTX3090; batch 4; length 2048
FlatQuant [39]	75.0% (ideal)	+5.7	−2.32 ^{F6}	2.30× prefill; 1.76× decode
W4 A4 KV4; GPTQ				L2-7B; RTX3090; batch 64
FPTQuant [57]	75.0% (ideal)	+11.0	−3.35 ^{F6}	In/out 2048/256; vs. FP16
W4 A4 KV4; GPTQ				2.8–3.9× block prefill
Atom [41]	75.0% (ideal)	+11.9	−9.25 ^{Q5}	Model sweep; RTX3080Ti
W4 A4 KV4 g128				Batches 1/16; length 1024
QoQ (QServe) [15]	75.0% (ideal)	+3.7	−1.03 ^{Q5}	3.40× GEMM
W4 A8 KV4 g128				W4A4; batch 512; RTX4090
				5.39× max. throughput
				L2-7B; L40S 48GB; in/out 1024/512
AWQ (adapted) [15,36]	75.0% (ideal)	+6.6	–	Vs TensorRT-LLM FP16; max. batches
W4 A8 KV4 g128				–

Bold labels identify headings and configuration groups. ↓: decrease in the named resource or rate. Source status/versions: Supplementary Evidence S1. Quality sources: A6 [51]; D5/D6 [55]; F6 [57]; Q4c [28]; Q5/adapted AWQ [15]; SqueezeLLM [31]. All task means are zero-shot. A6/D6 use ARC-e/c, PIQA, HellaSwag, WinoGrande, and BoolQ; D5/Q5 omit BoolQ; F6 replaces BoolQ with LAMBADA; Q4c averages ARC-e/c, PIQA, and WinoGrande. L1/L2/L3: LLaMA/Llama 2/Llama 3; HF: HuggingFace; NR: not reported; –: unavailable.

Quantization changes numerical precision, while pruning changes which connections, structures, or computations remain. Section 4 examines how that choice affects capability retention and the conversion of sparsity into executable savings under the intended workload.

4. Pruning and Sparsity

Sparsity granularity determines which information can survive and how the result can run. Individual-weight selection permits fine control, structural deletion produces smaller dense operators, and input-dependent selection changes active computation. The evidence below connects these choices to quality, preparation, and execution.

4.1. Sparsity Targets, Granularity, and Accounting

Static weight pruning applies a binary mask M to $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ as in Equation (3):

$$W_s = M \odot W, \quad s_W = 1 - \frac{\|W_s\|_0}{d_{\text{out}}d_{\text{in}}}, \quad (3)$$

Here, $\odot$ is element-wise multiplication and $\|\cdot\|_0$ counts nonzeros; s_W is the zero fraction of this matrix. Unstructured masks choose individual entries, semi-structured masks constrain local groups, and structured deletion removes channels, heads, or layers together with their dependent parameters [58–60]. We use $N:M$ to mean at most N nonzeros in each designated group of M entries. The grouping direction remains part of the pattern. Figure 4b illustrates how irregular zeros, a local 2:4 mask, and channel removal impose different structural constraints. A zero fraction within selected linear layers also differs from the fraction of all model parameters removed.

Input-dependent sparsity changes the selected subset at a token, sequence, or batch time scale [61–63]. It can leave parameter storage intact while reducing activity. Static zeroing also leaves dense allocation intact unless the representation changes: sparse storage adds indices or masks to surviving values, whereas structural deletion yields reduced dense dimensions. Thus, the key distinction is between reducing stored capacity, reducing the active subset, and changing the shape of the computation.

4.2. Unstructured Weight Pruning and Error Control

Pruning one weight changes a layer's output in a way that other weights may partly compensate. SparseGPT uses calibration inputs and second-order information to make such compensation part of pruning [58]. The underlying reconstruction problem can be written as Equation (4):

$$\min_{\hat{W}} \|WX - \hat{W}X\|_F^2 \quad \text{subject to} \quad \|\hat{W}\|_0 \leq k, \quad (4)$$

with calibration inputs in the columns of X and nonzero budget k . Approximate updates, shared column processing, and adaptive mask selection make the problem manageable. The calibration-input Gram matrix supplies information about this local loss, rather than the full training Hessian. Surviving weights can consequently change even though the procedure does not retrain the model end to end.

Wanda spends less preparation effort on correcting values. It ranks weights within each output channel using $|W_{ij}| \|X_{j,:}\|_2$, combining weight magnitude with the magnitude of the corresponding input feature [64]. The selected weights keep their values. This contrast separates two uses of activation evidence: Wanda uses it to decide what to remove, while SparseGPT also uses it to adjust what remains. Both depend on the calibration distribution.

The pruning budget can be allocated across layers as well as within them. OWL assigns non-uniform layer-wise ratios using outlier distributions and can accompany an existing pruning rule [65]. ALPS instead changes the reconstruction solver: operator splitting optimizes support and surviving weights, followed by conjugate-gradient refinement [66]. Layer allocation and sparse reconstruction therefore provide complementary controls over a model-wide budget.

The Wanda experiments distinguish the effect of weight selection from that of sparsity constraints. For LLaMA-7B on WikiText, dense perplexity is 5.68. At 50% unstructured sparsity, magnitude pruning gives 17.29, SparseGPT 7.22, and Wanda 7.26. Under a 2:4 constraint, SparseGPT and Wanda give 11.00 and 11.53, respectively (Table 3 in [64]). These are the source table's results for a fixed calibration set; Appendix D.2 of the Wanda paper [64] separately reports means and standard deviations across five calibration seeds. Activation-informed selection and compensated reconstruction both improve substantially over the tested magnitude rule. The higher perplexities under 2:4 show a separate cost: equal nominal sparsity can preserve different information when local constraints restrict which weights may survive.

4.3. Semi-Structured Sparsity and Learnable Masks

Semi-structured patterns make retained positions predictable for supported kernels while forcing nearby weights to compete. RIA combines relative weight importance and activation information with channel permutation [67]. Corresponding input transformations preserve the original map before pruning; the reordered groups then change which weights can survive.

MaskLLM freezes weights and optimizes a distribution over admissible masks through Gumbel–Softmax sampling, then uses a discrete mask at inference [60]. This adds end-to-end preparation while retaining the pattern constraints required by the execution kernels.

4.4. Structured Pruning and Compact Dense Models

Coarser deletion sacrifices selection freedom to produce smaller dense matrices. LLM-Pruner identifies interdependent structural groups, estimates importance with gradient information, and recovers quality through low-rank adaptation [59]. Removing a feed-forward channel, for example, must modify the projections on both sides consistently. All parameters in that group share the deletion decision. The advantage is that the resulting dimensions can expose savings to conventional dense computation.

SliceGPT changes the basis before making this coarse decision. Computational invariance around normalization and principal-component transformations allow it to slice low-contribution rows and columns in transformed coordinates [68]. The transformation preserves the mapping; dimension removal creates the approximation. FLAP takes another route by using activation fluctuations to estimate recoverability, standardizing importance scores for adaptive selection, and constructing bias compensation from baseline activations [69]. These methods improve structural deletion through, respectively, the coordinates being removed and the correction applied afterward.

Depth pruning targets the sequential chain of blocks. ShortGPT estimates block influence through changes in hidden-state direction and removes low-influence blocks [70]. The operation reduces both retained parameters and the number of transformations traversed. Its selection proxy captures limited change on the observed inputs, however, so the retained capability is determined by the jointly shortened model rather than by an isolated block score.

Sheared LLaMA organizes pruning around a target architecture and a larger recovery budget. It learns masks across structural dimensions, continues pretraining after pruning,

and adjusts domain mixtures through dynamic batch loading [71]. Together, these examples distinguish direct structural selection, constructed compensation, and continued model adaptation as routes to a compact dense model.

4.5. Input-Dependent Activation and Contextual Sparsity

Deja Vu predicts active feed-forward neurons and attention heads, combining sparse computation with asynchronous execution [61]. Both prediction accuracy and the cost of selecting components affect the result.

TEAL selects activation thresholds and allocates sparsity through block-wise reconstruction without updating weights [72]. CATS thresholds gated-MLP outputs and supplies sparse kernels, evaluating both training-free operation and optional fine-tuning with fixed thresholds [73]. Comparisons must identify the CATS branch because recovery changes preparation cost and quality.

ProSparse changes the distribution more explicitly. It converts activations to ReLU and combines progressively scheduled sparsity regularization with activation thresholding [74]. The resulting inactivity is learned through adaptation. The reported sparsity belongs to the adapted model, making activation changes and their training cost part of the deployment choice.

GRIFFIN selects neurons from prompt activations and reuses the subset during generation [62]. Probe Pruning combines a small batch probe with historical information to select a batch-specific structure [63]. Longer reuse amortizes selection but assumes that the selected components remain relevant.

4.6. Calibration, Recovery, and Combined Compression

Calibration is the evidence from which many masks and thresholds are inferred. *Beware of Calibration Data for Pruning Large Language Models* reports that changing the calibration source can alter pruning outcomes and that simply adding samples does not eliminate every source mismatch [75]. This result bears directly on activation-informed scores and reconstruction: greater precision in estimating one distribution need not repair its mismatch with the desired tasks.

Recovery can modify support, surviving values, or auxiliary updates. DSnoT repeatedly prunes and regrows connections using reconstruction error without conventional weight fine-tuning [76]; its dynamic mask changes occur during preparation. LaCo holds the mask fixed and sequentially optimizes surviving weights to align hidden states with the dense reference [77]. LoRAprune uses low-rank adaptation for importance estimation and iterative recovery alongside structured pruning [78]. REP-LIE also estimates importance from LoRA gradients and combines progressive structural pruning with updates restricted to the low-rank adaptation parameters [79]. Its proposed pruning procedure incorporates a ranking-stability criterion and resource constraints. Evaluations on LLaMA-7B and Mistral-7B extend the study beyond encoder models. Here, low-rank adaptation supplies an importance signal and a recovery mechanism; structural removal determines the compact model. This example therefore connects the cost of preparing a pruned model to the quality retained after recovery. These mechanisms act on different parts of the approximation, making the optimized object more informative than a binary fine-tuning label.

Combined methods distribute recovery across representations. Minitron couples architecture pruning to distillation-based retraining [80]. OATS constructs sparse-plus-low-rank weight approximations using activation-derived outlier information, while STBLLM combines structured sparsification with binarization and differentiated weight treatment [81,82]. OATS retains an additional low-rank path; STBLLM changes the precision of retained

weights. Their mechanisms therefore extend the pruning decision itself. Section 3 explains the corresponding representation choices, and the later synthesis considers their combinations.

4.7. Sparse Execution and Deployment Conditions

Fine-grained sparsity creates irregular access even when it reduces the weight payload. Flash-LLM resolves this tension by loading compressed sparse weights, reconstructing dense tiles on chip, and executing arithmetic with Tensor Cores [83]. It accepts dense arithmetic within the tile to retain an efficient computation path after reducing external weight traffic.

PowerInfer links input-dependent activity to placement. Frequently activated neurons reside on the GPU and less frequently activated neurons on the CPU; predictors and neuron-aware operators coordinate their use [84]. This arrangement targets models whose full storage exceeds GPU capacity. Its success depends on concentrating much of the activity in the GPU-resident subset while managing CPU work and communication.

SPDP combines static sparse storage with activation-dependent skipping. A column-oriented bitmap represents the sparse weights, and separate kernels accommodate prefill and decoding [85]. The decoding path can skip work using current activations, while the prefill path addresses its different operator shapes. Its model-level decoding results and prefill kernel measurements consequently describe different parts of that coordinated design.

These systems connect the selection time scale to reuse. Across a batch, the union of active neurons can exceed the subset required by one sequence, weakening sparsity in shared computation [61,84]. GRIFFIN's longer-lived selection and Deja Vu's contextual prediction also incur different selection frequencies. The useful question is therefore where reuse occurs: in weight access, in an active subset, or in a device placement decision.

4.8. Pruning Trade-Offs and Open Issues

Table 4 collects 21 source-relative comparisons for Llama 2-7B, grouped by unstructured, 2:4, and structural pruning. Budgets describe zero weights or removed backbone parameters; deployment entries report execution savings. The structural comparison evaluates quality immediately after pruning, before recovery training.

Table 4. Pruning and sparsity: budgets and quality changes for Llama 2-7B relative to each source's dense baseline, alongside independent deployment results.

Quality/Pruning-Budget Case				Independent Deployment (S)
Method	Resource Budget	Δ PPL (%)	Δ task (pp)	Result/Tested Configuration
Unstructured (reported budgets)				
Magnitude	50% zeros	+190.8	−8.57 ^{P7}	–
SparseGPT [58]	50% zeros	+27.1	−3.47 ^{P7}	1.82× CPU inference OPT-2.7B; DeepSparse; i9-7980XE 18 cores; 400-token batch; vs. Dense
Wanda [64]	50% zeros	+25.4	−3.47 ^{P7}	–
Pruner-Zero [86]	50% zeros	+22.3	−0.84 ^{P7}	–
RIA [67]	50% zeros	+27.6	−4.05 ^{O7}	1.50× inference L2-7B; 2:4; 2 A100s; batch 64; in 128
OptiShear [27] ^v	50% zeros	+26.0	−2.25 ^{O7}	1.35× speedup (CUTLASS sim.) L2-7B; 2:4; RTX A6000
DSnoT [76]	70% zeros	+1420.1	−26.77 ^{D7}	–
ALPS [66]	50% zeros	+25.6	–	–
Semi-structured (2:4)				
SparseGPT [58]	50% zeros	+98.6	−8.77 ^{P7}	1.54–1.79× matmul kernel OPT-175B shapes; 2:4; 2048 token rows
Wanda [64]	50% zeros	+115.2	−10.96 ^{P7}	1.24× speedup L1-7B; 2:4; A6000
Pruner-Zero [86]	50% zeros	+105.5	−7.69 ^{P7}	–
ALPS [66]	50% zeros	+82.6	–	–

Table 4. Cont.

Quality/Pruning-Budget Case				Independent Deployment (S)
Method	Resource Budget	Δ PPL (%)	Δ task (pp)	Result/Tested Configuration
Structured (25% backbone removed; no recovery)				
ShortGPT [70]	25% removed	+357.2	−11.28 ^{T8}	Throughput ↑ 16.5% L2-7B-GPTQ; 25% layers removed Baseline also GPTQ-quantized
LaCO+ [70,87]	25% removed	+268.0	−12.49 ^{T8}	– 1.33× layer matmul
SliceGPT [68]	25% removed	+228.9	−6.56 ^{T8}	L2-7B; A100 80GB; length 2048 Token latency ↓ 12% L2-70B; A100 40GB; batch 1; out 128 Dense: 4 GPUs; sliced: 3 GPUs
Wanda-sp [64,69]	25% removed	+79.9	−8.04 ^{T8}	–
LLM-Pruner [59]	25% removed	+125.8	−13.87 ^{T8}	Latency ↓ 15.5%; mem. ↓ 19.5% L1-7B; 20%; A5000; in 64
ZipLM [88]	25% removed	+46.3	−5.37 ^{T8}	2.10× throughput GPT2-124M; V100; batch 16; length 1024
OSSCAR [89]	25% removed	+45.7	−5.97 ^{T8}	2.00× inference OPT-2.7B; V100
FLAP [69]	25% removed	+62.3	−8.60 ^{T8}	Throughput ↑ 31.2%; mem. ↓ 24.7% L1-7B; 20%; A100
Tyr-the-Pruner [29]	25% removed	+46.7	−3.32 ^{T8}	TTFT ↓ 22.1%; decode throughput ↑ 15.2% Llama 3.1-8B; MI250; in/out 2048/2048 Batch NR; vs. Dense

Bold labels identify headings and configuration groups. ↑/↓: increase/decrease in the named resource or rate. Source status/versions: Supplementary Evidence S1. Quality sources: P7 [86]; O7 [27]; D7 [90]; T8 [29]; ALPS [91]. P7/O7/D7 are source-specific seven-task zero-shot means; T8 adds five-shot MMLU. Backbone budgets exclude embeddings and the output head. LaCO+/Wanda-sp are source-adapted variants. OptiShear reports WikiText; other sources specify WikiText-2. ^v OptiShear uses arXiv v2. L1/L2: LLaMA/Llama 2; NR: not reported; –: unavailable.

In the Pruner-Zero-source comparison, Wanda's 50% unstructured configuration raises perplexity by 25.4%, whereas its 2:4 configuration raises it by 115.2%. SparseGPT exhibits the same direction within that source. The mask constraint thus carries a quality cost that accompanies its execution structure. The 70% DSnoT row evaluates a more aggressive budget; the structural rows instead test immediate quality after backbone removal.

Pruning consequently involves selecting a surviving structure and a recovery procedure. SparseGPT's CPU and sparse-kernel results and FLAP's compact dense model illustrate distinct execution options. The RIA and OptiShear runtime cases use 2:4 patterns, whereas their quality rows use unstructured masks; ShortGPT's throughput case starts from a GPTQ-quantized model. These cases identify the additional configuration choices needed after specifying a zero fraction.

After selecting a compact structure, a further question is how to retain or recover its capabilities. Section 5 examines teacher supervision as a means of training compact students or recovering compressed models, with its associated preparation costs.

5. Knowledge Distillation and Compact Models

Knowledge distillation supplies supervision through which a compact student acquires or recovers teacher behaviour. Its resource effects arise from the student architecture, numerical representation, and generation procedure. Three questions therefore organize this section: what information is transferred, how training contexts and supervision are constructed, and which models remain in deployment.

5.1. Transfer Targets and Teacher Access

The transfer target determines the information required from the teacher. Generated responses provide sequence supervision and can be retokenized for another student.

Distribution matching requires output scores over suitable corresponding events; a few returned token probabilities provide less information than a complete vocabulary distribution. Hidden-state or attention matching additionally requires internal activations and may need projections between representation spaces. Specifying these requirements directly makes the feasible methods clearer than the inconsistently used labels black-box and white-box.

Sequence-Level Knowledge Distillation transfers teacher-decoded translation outputs; TinyBERT matches embeddings, states, attention, and predictions; MiniLM matches attention distributions and value relations [92–94]. These earlier translation and encoder studies establish transfer mechanisms for the subsequent discussion of autoregressive LLMs.

The student’s starting point determines what that supervision adds. An independently pretrained student brings capabilities learned before distillation, whereas a student initialized by pruning inherits selected teacher parameters. Task-specific imitation, instruction adaptation, and compact-model pretraining consequently represent different transfer scopes. The contribution of distillation is assessed against a stated student baseline, such as its initialization or a same-size SFT model, alongside the final student configuration.

5.2. Distribution Matching and the Capacity Gap

Let $p_T(v | c)$ and $p_S(v | c)$ denote teacher and student next-token probabilities given a common context c and vocabulary $\mathcal{V}$. Two common Kullback–Leibler (KL) divergences are defined in Equation (5):

$$\begin{aligned} D_F(c) &= \sum_{v \in \mathcal{V}} p_T(v | c) \log \frac{p_T(v | c)}{p_S(v | c)}, \\ D_R(c) &= \sum_{v \in \mathcal{V}} p_S(v | c) \log \frac{p_S(v | c)}{p_T(v | c)}. \end{aligned} \quad (5)$$

Forward KL penalizes missing teacher probability mass; reverse KL penalizes student mass where the teacher assigns low probability. When a limited-capacity student cannot reproduce the full teacher distribution, these tendencies change which discrepancies receive attention. Their effect on generation also depends on the contexts visited during training and on optimization [95,96].

MiniLLM combines reverse-KL minimization, policy gradients, teacher-mixed sampling and length normalization [95]. For LLaMA-13B-to-7B transfer, Table 1 reports SelfInst ROUGE-L of 20.2 for conventional KD, 20.8 for SeqKD and 23.2 for MiniLLM, against 23.4 for the teacher. Evaluation uses temperature-1 sampling and averages five generations with different random seeds. MiniLLM adds a three-epoch supervised initialization chosen by validation loss before policy optimization. Final checkpoints for MiniLLM and the baselines are selected by validation ROUGE-L. The result compares complete training procedures; isolating KL direction requires controlling these preparation and optimization choices.

Other objectives change how the capacity mismatch enters optimization. DistiLLM compares a distribution with a teacher–student mixture through skew KL variants [24]. DistiLLM-2 assigns skew forward and reverse objectives to teacher- and student-generated responses, respectively, and combines them with adaptive weighting and data curation [97]. Successful teacher continuations and imperfect student continuations thus contribute differently to learning. These methods change both the discrepancy being minimized and the examples on which it is evaluated.

TAID adaptively interpolates detached student and teacher logits, while Concrete Score Distillation matches relative vocabulary-logit differences [98,99]. They change target difficulty or the information matched before normalization. Together with skew divergences, these choices can alter training while leaving the deployed student architecture fixed.

5.3. Training Contexts and Supervision Construction

A student trained only on teacher continuations encounters different prefixes when it conditions generation on its own mistakes. GKD obtains teacher feedback on student-generated sequences and studies the interaction between divergence choice, quality, and diversity [96]. Its autoregressive T5 experiments provide evidence for this context-selection mechanism. DistiLLM combines adaptive use of student responses with off-policy reuse, while DistiLLM-2 collects responses in batches before each training epoch [24,97]. Reuse reduces repeated generation but increases the distance between stored trajectories and the evolving student; reported training acceleration concerns this preparation process.

Speculative Knowledge Distillation places teacher intervention inside trajectory construction: the student proposes tokens and the teacher replaces proposals that fail its acceptance criterion [100]. The resulting contexts retain student behaviour while limiting severely degraded continuations. Once training finishes, this student can generate independently.

Teacher guidance can also be moved into a reusable corpus. MiniPLM selects pretraining examples using the difference between teacher and reference-model log-likelihoods, then trains students from scratch on the refined data [101]. Teacher evaluation is performed offline and can support multiple students.

Text-based supervision follows the objective in Equation (6):

$$\mathcal{L}_{\text{text}} = -\mathbb{E}_{(x, \tilde{y}) \sim \mathcal{D}_T} \sum_{t=1}^{|\tilde{y}|} \log p_S(\tilde{y}_t | x, \tilde{y}_{<t}), \quad (6)$$

Here, $\mathcal{D}_T$ contains prompts and teacher responses. Unnatural Instructions constructs and reformulates instruction–response pairs, WizardLM expands instruction complexity through Evol-Instruct, and BalDistill allocates teacher queries across long-tailed tasks [102–104]. Their preparation budgets include scoring, generation, and filtering, with accepted tokens and task coverage distinguishing corpora of similar sample counts.

Figure 5 contrasts two deployment roles under teacher supervision: an independent student and a distilled draft paired with a retained target. The deployed role determines whether larger-model execution remains part of inference.

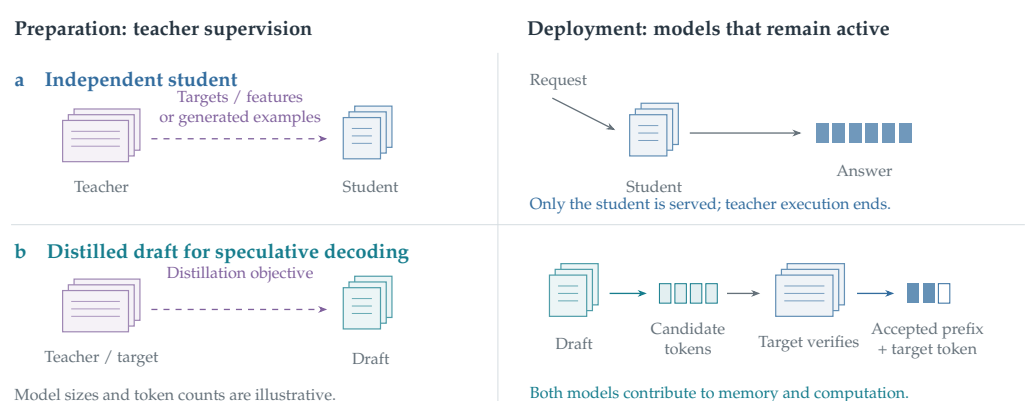


Figure 5. Teacher supervision and inference participation are separate choices. (a) An independent student uses teacher targets, features, or generated examples during preparation and serves requests without teacher execution. (b) A distilled draft proposes tokens that a retained target verifies during speculative decoding. The schematic output shows an accepted prefix as filled tokens and a target-generated continuation as an outlined token. The two deployment roles require different memory and computation accounting. Model glyph sizes are schematic. Relevant examples are Minitron and DistillSpec [80,105].

5.4. Instruction and Reasoning Transfer

Instruction transfer uses the constructed corpus to shape responses to user requests. Reasoning transfer extends that supervision to intermediate explanations, introducing another choice: whether the student must produce those explanations during inference. Fine-tune-CoT trains smaller models on diverse teacher-sampled rationales [106]. Symbolic Chain-of-Thought Distillation likewise transfers multiple reasoning paths and evaluates greedy decoding separately from self-consistency [107]. Multiple paths broaden supervision, while voting over several deployed responses spends additional inference computation.

Distilling Step-by-Step uses separate prefixes for labels and rationales, allowing prediction without generating a rationale. Socratic CoT uses a decomposer and solver that exchange subquestions [108,109]. SCOTT combines contrastive teacher decoding and counterfactual student training to test answer dependence on rationales [110]. This tests consistency under rationale changes; factual validity remains a separate property. Its teacher decoding requires scores beyond text-only access.

DeepSeek-R1 transfers reasoning to Qwen- and Llama-based students through supervised fine-tuning on approximately 800,000 examples, without an additional RL stage for those distilled students (Supplementary Sections 2.4.3 and 6 of [111]). Supplementary Table S13 of that study reports AIME 2024 pass@1 of 72.6 for DeepSeek-R1-Distill-Qwen-32B and 47.0 for Qwen2.5-32B-Zero, the direct-RL comparator trained for over 10,000 steps. This comparison holds student size at 32B while changing training data, teacher-derived supervision and preparation resources. The higher score therefore supports the complete distillation pipeline in this setting; a comparison of training objectives or total cost would also need to control those inputs and account for teacher creation.

Reasoning supervision can also increase response length. Skip-Thinking uses chunk-wise rationale training followed by a procedure that skips less important intermediate chunks [112]. In its principal GPT-2 and T5 student settings, the compression target includes the externalized reasoning trajectory. Together with Step-by-Step and Socratic CoT, this illustrates three ways to deliver reasoning-related capability: rationale-free task prediction, shortened traces, and interactive decomposition. Answer quality, emitted tokens, and time to the answer resolve those differences more directly than parameter count alone.

Training for compact reasoning links distillation to request compression. TokenSkip uses shortened correct trajectories as supervision, while CoT-Valve and TALE's post-training route learn controls over generated length [8,113,114]. These objectives can reduce generation without reducing parameter count. Section 7.6 compares their deployed controls and actual output lengths; Section 7.7 examines the accompanying accuracy–latency trade-off.

5.5. Heterogeneous Students and Deployment Roles

Changing the student tokenizer complicates Equation (5): vocabulary indices and token positions no longer identify corresponding output events. Universal Logit Distillation uses a Wasserstein-motivated comparison approximated with padded, sorted probability vectors, alongside token-level cross-entropy [115]. It transfers probability structure without an explicit shared vocabulary; sorting itself does not align token meanings. DSKD instead projects teacher and student hidden states into common output spaces and uses cross-model attention to align differently tokenized sequences [116]. The distinction is an access and alignment choice: DSKD learns correspondence using internal states, whereas ULD operates on probability structure. Training projections and alignment modules must be distinguished from the final student architecture.

Architectural transfer changes the computation more directly. The Mamba in the Llama initializes recurrent components from Transformer attention projections and progressively distills hybrids retaining some attention layers [117]. Attention and recurrent components

have different context-state requirements, so the resource change depends on their mixture and implementation, without requiring proportional parameter reduction. Any additional speculative-decoding acceleration belongs to a further deployment choice.

For standalone students, teacher execution ends after preparation (Figure 5). Minitron prunes a dense architecture and recovers capability through teacher-guided retraining; LLM-QAT trains a low-precision student on model-generated sequences [34,80]. Architecture or precision sets the resource constraint, while distillation supplies recovery supervision.

DistillSpec keeps the larger target in service: a distilled draft proposes tokens for speculative decoding, and the target verifies them [105]. Alignment can improve acceptance, but target and draft together determine memory and execution cost. Distribution preservation follows from the exact acceptance and correction procedure; lossy variants change the quality–latency trade-off. Thus, unlike the training intervention in Speculative Knowledge Distillation, target intervention continues on deployed requests. Draft latency, acceptance, and verification cost determine whether this auxiliary student use accelerates generation.

5.6. Quantitative Synthesis

Table 5 asks how much teacher supervision improves a chosen compact student. Each source fixes the student family and size. Parameter savings describe the teacher-to-student size difference; quality changes measure gains or losses relative to the same-size SFT student. This comparison evaluates the training pipeline within a fixed student capacity.

Conventional KD falls below the OpenLLaMA2 SFT student on all five reported tasks, while MiniLLM, GKD, and DistiLLM improve the reported means. In the Qwen2 comparison, DistiLLM-2 improves mean win rate over SFT by 28.36 percentage points, compared with 26.02 for DistiLLM. The student size reduction is shared within each block, so the differentiating result is the capability obtained through training. Distillation can therefore support a chosen compact architecture or recover another compression intervention, as illustrated by Minitron and LLM-QAT above.

Distillation specifies how a compact model learns; its capacity also depends on the representation it uses. Section 6 examines low-rank factorization as a complementary route that allocates capacity across matrix subspaces, with consequences for quality, storage, and execution.

Table 5. Distillation: nominal model-size savings relative to the teacher, and quality gains or losses relative to the same-size SFT student.

Method	Δ Dolly	Δ Self-Instr.	Δ Vicuna	Δ Super-Nat.	Δ Unnatural
OpenLLaMA2: 7B $\rightarrow$ 3B; nominal parameters $\downarrow$ 57.1%; Δ ROUGE-L points vs. SFT					
KD [118]	−4.16	−0.40	−0.94	−1.35	−4.01
SeqKD [92]	−0.44	−0.69	+0.73	−0.22	−0.61
ImitKD [119]	−0.58	+1.28	+1.03	+2.22	+0.37
MiniLLM [95]	+2.77	+3.42	+4.17	+7.63	+7.16
GKD [96]	+1.19	+3.04	+2.33	+6.43	+3.40
DistiLLM [24]	+4.62	+3.87	+3.29	+8.36	+8.39
Method	Δ AlpacaEval	Δ Evol-Instruct	Δ UltraFeedback	Δ Mean	
Qwen2: 7B $\rightarrow$ 1.5B; nominal parameters $\downarrow$ 78.6%; Δ win rate (pp) vs. SFT					
DistiLLM [24]	+15.24	+26.61	+36.25	+26.02	
DistiLLM-2 [97]	+18.82	+29.13	+37.12	+28.36	

Bold labels identify headings and configuration groups. $\rightarrow$: teacher-to-student size; $\downarrow$: parameter reduction. Source status/versions: Supplementary Evidence S1. Quality sources: OpenLLaMA2 [24]; Qwen2 [97]. Positive values indicate gains over SFT. OpenLLaMA2 uses five-seed means; Qwen2 means retain source-reported averages. Qwen2 win rates compare against text-davinci-003 (AlpacaEval) or gpt-3.5-turbo (Evol-Instruct/UltraFeedback).

6. Low-Rank Factorization

Low-rank compression restricts a transformation to a smaller subspace. The central problem is deciding which directions the model can lose: directions that contribute little to weight reconstruction need not be unimportant to the activations encountered during generation. This has motivated three connected developments: objectives informed by data, unequal allocation of rank, and decomposition units that extend beyond individual matrices. Together, they explain why the same number of retained parameters can represent different amounts of useful information.

6.1. Representation, Rank, and the Compression Budget

Figure 4c illustrates the two-factor representation. For a weight matrix $W \in \mathbb{R}^{m \times n}$, a rank- r replacement takes the form in Equation (7):

$$W \approx AB, \quad A \in \mathbb{R}^{m \times r}, \quad B \in \mathbb{R}^{r \times n}, \quad \rho_{\text{par}} = \frac{r(m+n)}{mn}. \quad (7)$$

The retained parameter fraction ρ_{par} counts both dense factors and excludes bias and auxiliary storage. At equal precision, storage falls when $r < mn/(m+n)$. For a 4096×4096 matrix, $r = 1024$ retains half the parameters, whereas $r = 2048$ retains all of them across the two factors. Each rank increment costs $m+n$ parameters, so differently shaped projections incur unequal costs. Shared bases, quantized factors, and residuals require their own storage accounting.

Truncated singular value decomposition (SVD) minimizes the Frobenius reconstruction error of the decomposed matrix at a given rank. The choice of matrix is therefore consequential. Decomposing weights preserves their dominant directions; decomposing transformed weights or activations can emphasize directions important to actual inputs. Dobi-SVD develops this distinction alongside the storage limitations of conventional factorization [120]. Low-rank adaptation has a different starting point: adding adapters to an intact base model leaves its original storage requirement in place. Low-rank model compression replaces base matrices with compact factors; any retained recovery adapters also contribute to storage.

6.2. From Weight Reconstruction to Data-Aware Objectives

The effect of a weight error depends on the inputs that encounter it. For calibration activations X , this motivates Equation (8):

$$\min_{\text{rank}(\hat{W}) \leq r} \left\| (W - \hat{W})X \right\|_F^2. \quad (8)$$

Unlike unweighted reconstruction of W , this objective can tolerate errors along directions weakly represented in X while protecting directions that contribute strongly to layer outputs. Its information about importance comes from calibration data; rare or shifted inputs can emphasize a different subspace.

Methods differ in the signal used to express that importance. Fisher-weighted factorization assigns greater weight to task-sensitive parameters [121]. Its encoder and earlier language-model experiments provide historical motivation for unequal error weighting. ASVD instead incorporates activation statistics into the decomposition and searches layer sensitivities when choosing compression levels [122]. These approaches connect truncation to model use, but employ different approximations to the consequences of losing a direction.

SVD-LLM whitens the calibration inputs to align truncation with local output error [123]. If $XX^\top = SS^\top$ is full rank, $S^{-1}X$ has orthonormal rows; truncating WS and

mapping back through S^{-1} then solves the transformed reconstruction problem. Sampling, covariance conditioning, and regularization affect practical whitening.

The full SVD-LLM procedure subsequently updates each factor using low-rank adaptation, holding the other fixed and then reversing their roles [123]. Merging these updates preserves factor shapes. Whitening selects the initial subspace; factor-wise training recovers behaviour within that representation, adding preparation work beyond the whitening-only variant.

SVD-LLM illustrates the quality cost of reducing weight storage. Its nominal compression ratio denotes parameter reduction: the 20% and 40% settings retain 80% and 60% of parameters, respectively. For LLaMA-7B, the source lists weight memory of 10.2 and 7.76 GB and WikiText-2 PPL of 7.73 and 9.27, compared with 13.5 GB and 5.68 for the uncompressed reference (Table 1 in [123]). These results use the full procedure, including factor updates with 50,000 Alpaca examples; the whitening-only variant is reported separately. The memory values use the source's reported units and account for a different quantity from the nominal parameter budget.

6.3. Allocating Rank Across Unequal Modules

Once the objective reflects output sensitivity, the next question is where to spend rank. Uniform compression assigns the same retained fraction to projections with different dimensions and functions. SVD-LLM V2 uses truncation loss to distribute compression non-uniformly and refines singular-value truncation [124]. This moves the budget decision from independent matrix size reduction toward balancing local losses across the model. The resulting allocation and the reconstruction objective are coupled: changing the importance measure can change which modules receive capacity.

Dobi-SVD learns truncation positions through differentiable optimization, introduces stable SVD backpropagation, and reconstructs weights from truncated activation information [120]. Incremental principal component analysis addresses the memory required for reconstruction. LLRC takes a related optimization route by learning masks over singular values, using representation matching while freezing the original model parameters [125]. Both make subspace selection an optimized component of compression. Their preparation budget includes rank-search optimization, which can use backward passes and teacher activations even with frozen model weights.

Rank search selects surviving directions, while recovery repairs the selected representation. Comparing allocation rules therefore requires the same final budget and recovery effort. Sensitivity-based allocation can assign additional rank to a difficult matrix by reducing capacity elsewhere.

6.4. Modular Decomposition and Cross-Layer Sharing

Choosing the unit of decomposition changes what redundancy can be removed. MoDeGPT groups matrix pairs into functional modules and treats their nonlinear structure explicitly [126]. It uses Nyström, column–row, and singular value decompositions for different approximations, folding transformations into reduced matrices. This retains a module's functional relationships that independent matrix reconstructions can overlook. A nonlinearity between two maps constrains which operations can be folded together, so modular decomposition also determines the shape of the computation left after compression.

Basis Sharing exploits a different relationship: selected matrices across layers can use a common basis with individual coefficients [127]. Activation information guides the approximation, and the basis is stored once. The saving grows through reuse of the stored representation, subject to how well the chosen layers fit the common subspace. This differs from reducing every matrix independently, where each matrix carries its own basis cost.

A shared basis reduces parameter storage but may still be applied separately to the hidden states at each layer. MoDeGPT reduces module dimensions, whereas Basis Sharing reuses stored parameters; the two savings need not produce the same reduction in execution cost.

6.5. Low-Rank Compression for Mixture-of-Experts Models

Mixture-of-experts (MoE) models extend allocation to a collection of conditionally active transformations. Many expert matrices remain resident even though routing activates only a subset for each token. MoE-SVD uses selective decomposition, shared factors, and expert importance to exploit this structure [128]. It considers expert-layer sensitivity and chooses which components to retain, combining approximation within experts with redundancy across them.

Here, the distinction between stored and active parameters becomes central. Compressing seldom activated experts still reduces the model's total footprint, but it does not reduce every token's computation by the same amount. Conversely, changing frequently selected components directly affects the executed path. Rank and sharing decisions therefore interact with routing and expert placement. The architectural extension preserves the underlying principle of sensitivity-based allocation, while requiring separate interpretation of total storage, active computation, and communication.

6.6. Quality and Execution

Table 6 separates three low-rank decisions: selecting useful directions, allocating rank across modules, and sharing stored bases. Their combination determines both the retained subspace and its storage cost.

Explicit factors execute B followed by A , adding an intermediate representation. Rank allocation must therefore account for matrix shapes and traffic as well as parameter count. Section 8.2 examines CALDERA's allocation between correction rank and factor precision. Parameter compression addresses the model's persistent representation. Long contexts and concurrent requests also create state and processing demands, motivating the treatment of input evidence, learned context representations, KV state, and generated reasoning length in Section 7.

Table 6. Representative low-rank approaches and the decisions that determine their quality and resource effects.

Approach	Compression Decision	Resulting Distinction
Weighted SVD; ASVD [121,122]	Weight task sensitivity or activation statistics	Important directions depend on the chosen data and sensitivity signal
SVD-LLM; V2 [123,124]	Whitening, factor-wise adaptation, and unequal truncation	Decomposition, recovery, and cross-module allocation have distinct roles
Dobi-SVD; LLRC [120,125]	Optimize truncation or singular-value selection	Frozen model weights can coexist with optimized compression variables
MoDeGPT [126]	Compress functional modules	Module structure determines which operations can be reduced or folded
Basis Sharing [127]	Reuse bases across layers	One stored basis can still require repeated computation
MoE-SVD [128]	Select, factorize, and share expert components	Stored capacity and per-token active capacity change differently

7. Token, Context, and KV-Cache Compression

Compression during a request determines both which evidence the model can use and how much computation it performs before answering. Input selection removes supplied text, learned context representations encode it into fewer positions, and KV methods change the state available to attention. Reasoning-length control reduces the generated trajectory. These interventions have different information losses and execution requirements, even when each reports a smaller token count. This section follows their position in the request lifecycle and the capabilities that their compressed representations must preserve.

7.1. Compression Objects and the Request Lifecycle

Four objects organize the comparison: input text, learned context representations, stored KV state, and the generated reasoning trajectory. Hard prompt compression produces a shorter discrete text sequence through selection or rewriting. Soft compression produces continuous representations consumed as embeddings or cached activations. The hard/soft distinction used in prompt-compression reviews therefore concerns the form of the compressed input [14]. A learned gist can itself be stored in a KV cache; these object and mechanism categories overlap.

Figure 6 locates the four routes across context preparation, target prefill, and generation. Shorter input text reduces target prefill work; learned context can be encoded offline for reuse or during a request. KV methods operate during prefill or generation. Eviction after prompt encoding preserves the original prefill cost while reducing later state and attention access. Shorter output reduces autoregressive steps. These routes can be combined.

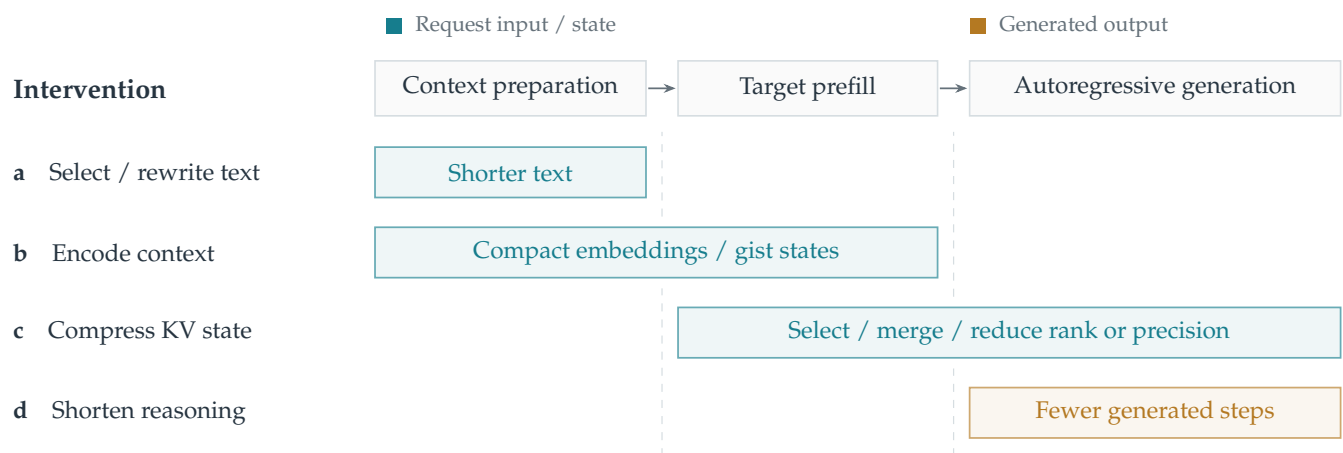


Figure 6. Intervention stages for four request-compression routes: (a) select or rewrite text, (b) encode compact context representations, (c) compress stored KV state, and (d) shorten generated reasoning. Horizontal spans indicate applicable stages. Learned context can be prepared offline or during request encoding. The routes can be combined.

Access requirements depend on the compressed object. Text selection uses the target's text interface; learned context needs compatible embeddings or cached representations; KV changes require cache or attention access. Output-length control uses prompting or model adaptation. Evaluation pairs retained facts, recall, and answer correctness with complete request cost, including compression, loading, selection, and budget estimation where used. Offline encoding and reusable storage are recorded separately.

Token counts must retain their meaning across these routes. A text token, a continuous input position and a cached position have different storage and access requirements. Equation (2) additionally depends on dimensions and precision. Reducing the initial prompt does not by itself reduce the KV state added by each new generated token; shorten-

ing generation reduces the number of such additions. Consequently, input length, retained state, and output length belong in separate comparison fields.

7.2. *Input Text Selection and Summarization*

Text compressors differ in how they estimate what can be removed. Selective Context ranks lexical units using self-information, treating predictable content as a candidate for deletion [129]. LLMingua combines budget control, iterative token compression and distribution alignment between the compressor and target, while LongLLMingua incorporates the question and document ordering [6,130]. The distinction is consequential: low surprisal is not the same as low relevance to a particular answer. Question-dependent selection can protect relevant details, but its output may need rebuilding when the question changes.

Learning a selection policy moves the decision into preparation. LLMingua-2 trains token classification from distilled compression supervision [131]. TACO-RL further optimizes a classifier with downstream task rewards and length constraints [132]. Its summarization, QA, and code-summary evaluations test whether task feedback improves the retained text at specified compression levels. The encoder processes 512-token chunks; testing compresses each chunk separately and concatenates the results. The reported quality therefore evaluates chunk-wise selection, leaving preservation of dependencies across substantially longer inputs to be tested. Training feedback and online token scoring are distinct cost terms.

Rewriting changes what the remaining positions can express. RECOMP studies both extractive sentence selection and abstractive summaries of retrieved evidence, including an empty summary when augmentation is unhelpful [133]. A summary can combine information dispersed across the source, whereas extraction preserves original spans. This adds a faithfulness question to relevance selection: does the rewritten evidence retain the relations required for answering? RECOMP's QA table reports evidence-text length separately from in-context examples, illustrating why the denominator of a compression ratio must be stated.

These methods expose shorter text through the target's ordinary input interface. Their evaluation should combine downstream quality with compressor cost and checks on retained instructions and facts. Shorter supplied text can reduce target prefill and initial KV occupancy; the resulting benefit belongs to the compressor–target procedure and depends on whether the compressed text is reused.

7.3. *Learned Context Representations*

Continuous representations replace text with a learned channel whose capacity and interface must be specified. Gist Tokens insert special positions after an instruction and modify attention masks so that subsequent tokens access the instruction through those positions [134]. Their KV activations can be cached and reused. AutoCompressor instead learns summary vectors over text segments and accumulates them for later processing [135]. The former establishes instruction compression; the latter extends the mechanism to segmented context and evaluates language modelling, in-context learning, and retrieval applications.

The location of the representation changes both learning and execution. ICAE adapts an encoder with LoRA to produce memory slots, using reconstruction and continuation objectives before instruction training [7]. The slots are consumed by the target LLM, so retaining its weights does not make the interface equivalent to ordinary text prompting. Gist and AutoCompressor likewise require models trained to use their representations. Encoding, representation storage, and target execution must therefore be counted separately. ICAE explicitly separates compression and generation in its timing study; AutoCompressor's retrieval experiments exploit precomputed summaries.

Retrieval embeddings offer a representation that may already exist before generation. xRAG trains a bridge from a dense document embedding to the LLM input space while freezing the retriever and language model [136]. Its experiments compress a retrieved document into one position. COCOM makes the number of context embeddings adjustable and adapts the decoder to consume multiple retrieved contexts; COCOM-light uses a smaller compressor [137]. Their comparison raises two linked choices: how much representational capacity to allocate per document, and which parts of the generator must adapt. Different numbers of retrieved passages and different decoder-training regimes remain part of that comparison.

Reuse can make representation preparation worthwhile, but its savings depend on the baseline. Gist's efficiency study compares compressed instruction caching with both no caching and full-instruction caching [134]. Gains over the latter are smaller because it already avoids re-encoding the instruction. A useful deployment comparison therefore retains the existing caching strategy and includes encoding, loading, and any query-dependent processing. Section 8.3 examines COCOM's generation and offline-storage trade-offs within its reported settings.

7.4. KV Position Selection and Budget Allocation

KV selection estimates which positions will remain useful to later attention. H₂O combines recent positions with heavy hitters selected from accumulated attention, whereas Scissorhands uses persistence of importance [138,139]. StreamingLLM protects initial attention sinks and a recent window to sustain bounded-state generation [140]. Sink positions stabilize attention even when their content is unrelated to a later answer. Stable streaming and preservation of detailed historical evidence are therefore separate capabilities.

Allocation adds structure to the retention rule. FastGen profiles attention to assign head-dependent policies; SnapKV uses an observation window at the end of the prompt to select clustered important positions before generation; PyramidKV redistributes capacity across depth [4,141,142]. These policies differ in the signal used, when it becomes available and where the budget is spent. SnapKV compresses an already encoded prompt, while ongoing policies update state as generation proceeds. Profiling and selection are recurring work when repeated per request.

Permanent eviction commits the model to a restricted history. A new query cannot directly access a discarded state without re-encoding or another recovery path. Fixed position counts become more restrictive as context grows; fixed retained fractions allow storage to grow. Protected sinks, recent windows, and newly generated positions also consume the nominal budget. Grouped-query attention introduces a further constraint: independent head policies must preserve shared KV structure or account for duplicated state.

7.5. KV Merging and Compact Representations

Merging allows removed entries to contribute through a retained approximation. MiniCache shares information between similar states in adjacent layers, separating direction and magnitude and preserving sufficiently distinct exceptions [143]. D2O incorporates discarded-token information through compensation and merging within dynamic layer allocation [144]. These methods exploit redundancy across depth or positions. Merging approximates the original attention computation using shared or aggregated state. Its effect on detailed recall depends on which distinctions the approximation preserves.

Low-rank representations preserve positions in fewer feature dimensions. Eigen Attention constructs reduced attention spaces; Palu factorizes KV projections, allocates ranks across head groups, and provides a fused execution path [5,145]. Palu can absorb value reconstruction into the output projection, while rotary positional embeddings require

different treatment on the key side. OjaKV updates its subspace online and protects selected full-rank states [146]. A changing basis may follow the current context more closely, while charging for maintenance and reconstruction.

Memory placement further changes the executed procedure. ShadowKV combines low-rank keys with value-cache offloading and selective KV reconstruction [147]. Reduced GPU residency can support larger batches, with transfer and selection costs attached to the result. Here, the relevant storage record includes state retained elsewhere in the memory hierarchy. Section 8.2 separately examines Palu’s capacity, quality, and timing settings.

Precision can be reduced alongside positions or dimensions. KIVI and KVQuant supply the numerical mechanisms described in Section 3.5, while Palu and MiniCache combine compact state with lower precision [2,5,43,143]. For homogeneous unshared caches, a useful accounting approximation is given in Equation (9):

$$\frac{M'_{\text{payload}}}{M_{\text{payload}}} \approx \alpha \frac{r_K b'_K + r_V b'_V}{d_h(b_K + b_V)}. \quad (9)$$

Here, α is the position fraction retained in both keys and values, d_h is the original dimension per KV head, r_K, r_V are retained dimensions, and b_K, b_V and their primed counterparts are original and compressed bits per element. Batch, layer, and KV-head counts remain fixed. Retaining half the positions and half the dimensions while changing FP16 to 4-bit storage gives $0.5 \times 0.5 \times (4/16) = 0.0625$, or 6.25% of the original payload. This ideal accounting excludes bases, metadata such as scales and indices, protected windows, and reconstruction buffers; shared states or unequal key/value selection require separate counting. Quality evaluation must also capture interactions between projection, quantization, and selection.

7.6. Reasoning-Output Compression and Length Control

Reducing the generated reasoning trajectory targets the number of autoregressive steps. The compression object is the trajectory produced to solve a task; the model can retain the same parameter count. This differs from shortening an explanation after generating it, which leaves the original decoding work unchanged. The relevant quality measure is the success of the resulting answer, accompanied by actual generated length and measured cost.

TokenSkip compresses correct training trajectories using token-importance scores, and it then fine-tunes the model on mixed retention ratios so it can directly generate a shorter chain of thought [8]. CoT-Valve controls reasoning length through a learned parameter-update direction, with training variants based on the trajectories of different lengths [113]. Both require training access, but they expose different controls over generation. TokenSkip evaluates accuracy, actual CoT length, and per-sample latency; CoT-Valve’s accuracy-per-computation-unit measure uses parameter count and token count as a proxy. Hardware timing and that proxy answer different efficiency questions.

TALE provides both an estimation-and-prompting route and a post-training route [114]. The former estimates a token budget before prompting the reasoning model, adding an estimation request. The latter internalizes budget awareness from targets constructed using offline budget search. Label-dependent search belongs to preparation, while a deployed estimator operates without the reference answer. These access and cost differences matter when choosing between a text-only interface and a trainable model.

A requested length is a control signal rather than a guaranteed output size. Evaluation should record actual tokens, correctness and termination rules, including any maximum-generation limit. TokenSkip’s settings illustrate how stronger shortening trades accuracy for latency in Section 7.7 and Table 7. Distillation supplies one way to construct efficient

reasoning behaviour, discussed in Section 5.4; this subsection concerns the trajectory that the deployed model actually generates.

7.7. Information Preservation and Comparative Synthesis

Table 7 groups KV-state, learned-context and reasoning-output compression, with a source baseline and resource unit for each group. KV percentages refer to the initial prompt: fixed capacity holds the retained count during decoding, while growing caches retain generated tokens as well. KIVI retains all positions; MiniKV Pyramid varies prompt allocation across layers at the stated mean. These policies follow the quality comparison in MiniKV; the separate deployment settings retain their own budgets. The context group uses full-context retrieval-augmented generation (RAG) as its quality reference. COCOM decode timing excludes retrieval and offline encoding. The output comparison follows TokenSkip, using greedy decoding, actual CoT token counts, and mean per-sample latency.

Table 7. KV, context, and output compression: resource savings and quality changes relative to the source baselines specified in each group.

Method/Configuration	Resource Saving	Quality Change	Deployment Result (S/P)
KV: Llama 2-7B-chat; reference = full KV16; quality = LongBench-13 points			
KIVI [2] KV2; all tokens; growing	KV scalar payload ↓ 87.5% [I]	−0.22	2.35–3.47× max-batch throughput (S) Llama 2-7B; KV2; A100
H ₂ O [138] KV16; 15% prompt; fixed	Prompt tokens ↓ 85% [B]	−1.70	1.86× vs. FlexGen (S) OPT-6.7B; 20% KV; A100 Batch 24; in/out 2048/2048
SnapKV [4] KV16; 15% prompt; growing	Prompt tokens ↓ 85% [B]	−1.59	About 3.6× decode (S) LWM; 16k to 2k KV; A100 Batch 2; out 512
Q-Hitter [148] KV4; 59% prompt; fixed	Prompt tokens ↓ 41% [B]	−2.18	4.09× vs. FlexGen (S) OPT-6.7B; KV4/20%; T4; in/out 512/512
MiniKV [45] KV2; 50% prompt; growing	KV size ↓ 86.3% [E] In/out 4096/512	−1.48	2.71× attention kernel (S) Vs standard attention; A100 40GB Length 2048; batch NR
MiniKV Pyramid [45] KV2; mean 50%; growing	Prompt tokens ↓ 50% [B]	−0.54	–
Context: Mistral-7B-Instruct-v0.2; reference = full-context RAG; quality = NQ EM (pp)			
ICAIE [7] 4:1; frozen decoder	Context slots ↓ 75% [B]	−38.70	3.29× incl. compression (S) LLaMA-7B; batch 8; A100 In/out 2048/128
xRAG [136] 128:1; frozen decoder	Context slots ↓ 99.2% [B]	−41.30	CUDA time ↓ 33.5% (S) Mistral-7B; BF16; A100
COCOM-light [137] 16:1; tuned decoder	GPU memory ↓ 20.4% [M] Context slots ↓ 93.8% [B]	−10.50	Decode latency ↓ 80.0% (S) NQ; FP16; A100 40GB; batch 16
COCOM [137] 16:1; tuned decoder	GPU memory ↓ 20.4% [M] Context slots ↓ 93.8% [B]	−5.80	Decode latency ↓ 80.0% (S) NQ; FP16; A100 40GB; batch 16
Output: Llama 3.1-8B-Instruct; reference = original generation; quality = GSM8K (pp)			
BeConcise [8] No numeric target	CoT tokens ↓ 24.3% [M]	−3.30	Latency ↓ 20.6% (P) RTX3090; batch 1
LC-Prompt [8] 50% target retention	CoT tokens ↓ 11.4% [M]	−2.50	Latency ↓ 16.6% (P) RTX3090; batch 1
Truncation [8] 50% target retention	CoT tokens ↓ 51.4% [M]	−79.20	Latency ↓ 50.5% (P) RTX3090; batch 1
TokenSkip [8] 50% target retention	CoT tokens ↓ 47.0% [M]	−8.00	Latency ↓ 43.0% (P) RTX3090; batch 1

Bold labels identify headings and configuration groups. ↓: reduction in the named resource or latency. Comparison sources: KV [45]; context [137]; output [8]. [M] measured; [B] nominal budget; [I] ideal scalar payload; [E] analytical estimate. P: quality and deployment from the same evaluation; S: separate deployment experiment. Negative quality changes indicate loss. NR: not reported; –: unavailable.

The KV group contrasts precision reduction with position removal in the MiniKV-source quality comparison. KIVI retains all positions, preserving their availability for later attention; eviction removes positions. Its deployment result concerns maximum-fitting-batch throughput, while MiniKV reports attention-kernel acceleration and slower prefill than FlashAttention in another setting. Retained history and the request stage being accelerated are therefore separate choices.

The context group also changes the interface and training requirements. ICAE and xRAG use frozen decoders in the displayed comparison, while COCOM tunes the decoder; xRAG uses one passage compared with five for RAG and COCOM. At 50% target retention, TokenSkip actually retains 53.0% of CoT tokens, reducing token count by 47.0% and latency by 43.0%, with an 8.0-percentage-point accuracy decrease. Truncation removes 51.4% of tokens but loses 79.2 points. Similar generated lengths can therefore preserve very different amounts of useful reasoning.

Task quality, grounding, and information preservation provide complementary tests. The evaluation by Łajewska et al. compares downstream performance with support for generated claims and reconstruction of compressed contexts, including entity preservation [18]. A fluent answer or a similar summary can coexist with missing details. Grounding scores depend on the LLM judge and its prompting, while reconstruction depends on the decoder. Checking sensitivity to judge and prompt changes, and agreement with human judgments, would strengthen these diagnostics of preservation.

The required information type explains important task differences. Deng et al. compare gist architectures primarily on Llama 3.1-8B under common training and evaluation conditions [19]. Stronger retrieval-augmented generation and long-document QA results coexist with poor precise synthetic recall; appendix experiments on Qwen2-7B and Llama 3.2-3B show similar task-dependent patterns. The detailed boundary, unexpected-information, and accumulation analyses remain tied to the main experimental setup. For KV eviction, The Pitfalls of KV Cache Compression evaluates Llama 3-8B and Qwen2.5-14B: degradation differs by instruction category, model and eviction policy [16]. LongBench supplies complementary long-context tasks [149].

Failure detection is a further decision layer. Belikova et al. test overflow detection in xRAG on examples that the uncompressed model already answers correctly [150]. Query-context representation probes are more informative than saturation statistics alone in that setting. Their prediction task concerns compression-induced failures on a filtered subset, which must remain distinct from full-dataset QA accuracy. The result motivates testing adaptive retention or fallback when a compressed representation is unlikely to support the current query.

Request compression targets three distinct costs: representing history, processing input evidence and generating an answer. The recall, grounding, and reasoning tests above diagnose losses at these different intervention points. Section 8 combines them with model compression in deployment selection.

8. Combined Compression and Deployment Trade-Offs

A deployment can combine a compact model with compressed history and shorter generation. The preceding sections establish the representation choices and their reported outcomes. This section connects those choices through shared budgets, interprets their distinct evidence, and applies them to a workload-specific selection sequence.

8.1. Complementary Mechanisms and Compression Order

Equation (10) gives an analytical template for combining quantized, low-rank, and sparse components:

$$\hat{W} = Q + AB + S, \quad (10)$$

where Q is the dequantized approximation of a dense component, AB a low-rank component, and S a sparse component. Methods with additive weight representations instantiate different subsets of these terms with their own constraints and objectives. LQER allocates a low-rank branch to quantization error and shapes its spectrum using activation-induced scaling [151]. CALDERA constrains both the backbone and factors to low precision under a calibration-aware reconstruction objective [25]. In these designs, a small rank need only represent a correction; approximating the entire weight matrix at that rank would be a different problem.

Sparse components introduce another distinction. SpQR retains quantization-sensitive weights as high-precision exceptions [152]. This protects selected information in an otherwise quantized matrix. LoSparse instead learns a low-rank component and progressively prunes a sparse residual; its DeBERTa and BART experiments establish that combination in their respective task and architectural settings [153]. OATS combines outlier-aware low-rank and sparse approximation, whereas STBLLM couples structured sparsity with binary representations [81,82]. Thus, sparse exceptions, a pruned residual and a structured zero pattern denote different retained information and execution requirements.

Equation 10 exposes allocation within one representation. Higher correction rank can be funded by lower factor precision, while sparse exceptions consume bytes otherwise available to the dense component. The CALDERA comparison below examines this allocation at a fixed reported average rate.

Other combinations assign methods to modules or preparation stages. SPQ applies factorization to attention projections, structured pruning to MLP components, and subsequent low-precision representation [154]. Minitron uses distillation to recover a structurally reduced model, while LLM-QAT uses teacher-generated data in quantization-aware training [34,80]. Here, supervision recovers capability within the resource constraint imposed by the final architecture or precision. Ordering matters because pruning changes the activations seen by a later quantizer, and quantization changes the errors that recovery must address. Published results for a pipeline support that final configuration; identifying a better ordering requires a controlled comparison of alternative sequences.

8.2. Interpreting Combined Evidence

Tables 3, 4 and 7 describe quality changes under representation or request compression. Table 5 measures teacher-guided gains over same-size students, with the size ratio referenced separately to the teacher. Independent deployment columns identify tested execution paths; paired measurements also connect quality and execution in one evaluation. These roles determine how the evidence supports a combined configuration.

Allocating a shared representation budget.

CALDERA compares two Llama 2-7B configurations with a 2-bit backbone and 2.4 average bits overall, without additional randomized Hadamard transform fine-tuning (Table 1 in [25]). Increasing correction rank from 64 to 256 while reducing factor precision from 16 to 4 bits changes WikiText-2 perplexity from 7.36 to 6.19 and C4 perplexity from 9.47 to 8.14. Zero-shot RTE accuracy, evaluated with the EleutherAI Language Model Evaluation Harness, changes in the opposite direction, from 66.4 to 60.6. These point estimates illustrate metric-dependent allocation; uncertainty for the reversal is unreported.

Selecting rank and precision therefore requires evaluating the intended capability alongside perplexity.

Separating capacity, quality, and timing.

Palu provides a worked example of scope separation (Table 2 and Figure 5 in [5]). Its Llama 2-7B WikiText-2 comparison at length 4096 reports perplexity 5.12 for the baseline and 5.77 for a configuration that halves the unquantized KV latent-state payload through low-rank projection and then applies 3-bit quantization. Cache sizes of 64.0 and 6.0 GB are illustrated at 128K tokens. A separate RTX 4090 benchmark uses 4-bit linear weights and measures attention modules or next-token decoding; its 2.59-fold decoding gain at 64K is relative to an estimated FP16-cache baseline after memory exhaustion. The quality test, long-context capacity illustration and decoding benchmark answer distinct feasibility questions.

8.3. Deployment Choices Under Quality, Capacity, and Latency Constraints

Table 8 screens candidates by the application constraint and available access. Quality limits follow from application failure costs or an acceptable change from the uncompressed reference; timing targets follow from the intended service objective. The sequence below applies these requirements to complete configurations.

Table 8. Candidate screening and minimum deployment checks. Compression changes a representation or generated trajectory; batching, allocation, and placement determine how it is executed.

Constraint	Candidates/Access	First Exclusion or Failure Check	Minimum Measurements
Weight residency	Weight quantization, structural pruning, factors; weight access and any required calibration/recovery	Exclude formats lacking a supported execution path or exceeding peak memory after metadata and buffers	Checkpoint bytes, peak device memory, required task scores, and loading cost
Long retained context	KV quantization, selection, merging or factorization; cache/attention access	Reject policies that lose delayed evidence or exceed capacity after residuals and reconstruction	Cache bytes vs. length, delayed-recall quality, prefill/decode latency, and peak memory
Interactive response time	Smaller dense models or supported quantized/sparse paths; runtime control	Exclude gains confined to unrelated operator shapes or batches; check online overhead	Time to first token, time per output token, tail latency, and quality under the arrival pattern
Offline throughput	Joint model/state compression plus batching or placement; scheduler and device/host access	Reject configurations violating quality, memory or batch-completion requirements	Throughput and completion time together, peak memory, transfer cost, and quality
Independent compact model	Distillation or recovery after structural reduction; trainable student and required teacher interface	Verify the student can run alone and retains the required capabilities	Student quality, parameters/bytes, memory, latency, teacher-generation, and training costs
Text-only access	Selection, summaries or output-budget prompting; text interface	Exclude methods needing inaccessible weights/cache; check preprocessing and estimator failures	Complete-request time/cost, evidence retention, correctness, and actual token counts
Repeated document retrieval	Learned context or retrieval embeddings; encoder/bridge and representation input	Check collection refresh cost, decoder compatibility, and multi-document grounding	Quality, encoding time, index size/loading, and request time including retrieval
Long reasoning output	TokenSkip/CoT-Valve or budget prompting; training/parameter or text access as required	Reject shortened traces that fail correctness; distinguish target from observed length	Answer accuracy, actual output length, latency, and preparation/estimation cost

Validation sequence.

1. Define input/output lengths, concurrency, required capabilities, and acceptable losses against a reference evaluated under the same protocol.
2. Use Table 8 to exclude inaccessible, unsupported, or over-capacity configurations, counting metadata and runtime buffers.

3. Check required task quality for each survivor; record preparation data, evaluation harness, and variability. Published evidence guides the shortlist.
4. Measure quality and resources jointly on the target runtime. Separate prefill, decoding, and whole-request costs; repeat timings and document the baseline.
5. Compare feasible configurations using the application's latency, throughput, or cost objective. Include preparation and refresh costs over expected reuse; reassess after workload or runtime changes.

The QServe study supplies a concrete capacity-then-throughput example. Fix an L40S-48GB and input/output lengths of 1024/512. For Llama 2-7B, its TensorRT-LLM comparison reports 1271 tokens/s with W8A8 and 681 with W4A16, each at its throughput-maximizing batch (Table 4 in the arXiv v3 version of [15]). Both pass the capacity screen; if both satisfy the required task quality, the reported throughput favours W8A8. Changing the target to Llama 2-70B excludes W8A8 through memory exhaustion, while W4A16 reaches 119 tokens/s and remains a candidate for quality validation. Thus, the same hardware and precision options lead to different candidate sets as model size changes. An interactive-latency objective would require timings under the intended arrival pattern.

Reusable context representations add preparation and storage costs to the request-time results in Table 7. COCOM reports index sizes of 6.06, 1.51, and 0.19 TB, with encoding times of 89, 77, and 73 h, for compression factors 4, 16, and 128 over 24 million passages on an A100-80GB (Table 4 in [137]). Its decode benchmark excludes retrieval and offline encoding. Collection updates and reuse frequency therefore determine how those preparation costs contribute to the deployed system.

LLMC supports configuration studies for applying these checks to a chosen runtime [50].

8.4. Cross-Method Findings for Deployment Selection

Capacity screening precedes speed selection. The QServe example changes the feasible precision set when the model grows on fixed hardware. For long requests, KV state can become the binding constraint after weights fit; Palu's capacity and timing cases illustrate the additional state representation and reconstruction choices. Model size and request shape jointly determine which compression target to examine first.

Allocation follows the required capability. CALDERA's rank-precision comparison favours different settings for perplexity and RTE in the reported point estimates. The pruning comparisons similarly attach a quality cost to local mask constraints. A storage target defines the available budget; task requirements determine which allocation remains useful.

Student construction and request compression are complementary. The distillation table compares learning outcomes within a chosen student capacity. KV and context methods instead change accessible history, while output control changes generation length. This separates two deployment questions: which compact model can perform the task, and which request information or steps it needs to do so. Evaluating the combined configuration determines whether the individual gains survive their interaction, motivating the capability-transfer, delayed-information, and complete-task evaluations in Section 9.

9. Open Challenges and Future Directions

The deployment choices above expose open questions about capability transfer, future uses of compressed history and interactions between compression stages. The following research directions specify evaluations that could resolve them.

9.1. Capability Retention Beyond Local Objectives

How well do preparation objectives predict downstream capability? SparseGPT and SVD-LLM reconstruct calibration computations; MiniLLM and GKD transfer teacher distributions [58,95,96,123]. The calibration study in Section 4.6 also identifies preparation-data selection as a variable to control [75].

A controlled comparison would vary preparation data and objectives at a fixed checkpoint, final resource budget, and recovery effort; distilled students also require controlled initialization and teacher access. A minimum capability profile should cover language modelling, knowledge, reasoning correctness, instruction following, long-context retrieval, robustness, and safety-sensitive behaviour, adding multilingual evaluation where required. Instruction-category differences in The Pitfalls of KV Cache Compression motivate this profile [16]; reasoning evaluations should also record generated length.

9.2. Preserving Information Across Long and Changing Contexts

A state compressor acts before all future uses of the history are known. SnapKV selects from prompt attention, StreamingLLM preserves a bounded history policy, and Palu limits the feature subspace used to represent states [4,5,140]. Each can preserve information useful in its tested setting while leaving a different vulnerability to later topic or instruction changes. The delay and change of purpose between storage and retrieval are therefore central evaluation variables alongside total context length.

LongBench supplies varied long-context tasks [149]. A complementary delayed-information test could (1) place a fact at a controlled position; (2) compress the history; (3) introduce a retrieval target after compression while varying retrieval delay at fixed total length; and (4) measure fact recovery, answer quality and cache cost under either fixed token counts or fixed retained fractions. The gist study identifies boundary, unexpected-information, and accumulated-loss patterns, while the information-preservation study separates task performance from grounding and retained entities [18,19]. Varying both information type and delay would test when protected state, periodic reselection, or recoverable storage is useful.

Adaptation can restore flexibility at an additional cost. OjaKV updates a subspace online, while ShadowKV combines compressed state, selective reconstruction, and placement across a memory hierarchy [146,147]. The question is how much adaptation or recoverable storage is needed for delayed information use under a fixed total budget. That budget includes basis maintenance, reconstruction, and history retained elsewhere in the memory hierarchy. The xRAG overflow study suggests a further question: can a query-dependent failure signal trigger a larger representation or retrieval fallback early enough to justify its cost [150]? Evaluation should include both detection errors and the task-level outcome of the resulting fallback policy.

9.3. Joint Compression Under Coupled Budgets

How should budgets and stage order be optimized together? LQER and CALDERA construct error corrections, SPQ assigns methods across modules, and Minitron recovers a reduced architecture through teacher supervision [25,80,151,154]. Earlier transformations change activations or capacity, potentially changing the best later allocation.

The decisive comparison is joint optimization against alternative stage orders and single-method baselines under the same final constraint. Stored bytes and preparation effort should be controlled separately: a larger correction or more recovery can improve quality without establishing a better interaction between mechanisms. Component ablations can test whether a residual corrects the intended error. The remaining systems question is whether fine-grained allocations are worth their additional formats, routing,

and kernel requirements. Allocation procedures should expose executable alternatives so that representational gains can be assessed alongside implementation complexity.

9.4. Executable Benefits and the Full Cost of Compression

Portability remains an open execution question for the representations used by QServe, FLUTE, and Flash-LLM [15,46,83]. Using infrastructure such as LLMC [50], a controlled study could vary hardware or kernels at a fixed representation and then vary batch size and context on a fixed runtime. This would measure how much of an observed gain survives changes in accelerator and workload.

Cost evaluation should vary request volume and update frequency. COCOM provides offline encoding and index costs, while TALE separates online budget estimation from post-training [114,137]. Amortizing teacher generation, calibration, and recovery over reuse would reveal when preparation pays off. Queueing belongs in tail-latency tests; energy requires measurement at a stated device or system boundary. Joules and monetary cost per successful task should include compression calls, tool use, retries, and failures.

9.5. Agent Tasks as a Future Evaluation Setting

Agent tasks extend the review's evaluation questions to interactions with tools and changing observations. ACBench reports task-dependent effects across action execution, workflows, long contexts, and interactive environments [17]. Agent Distillation transfers retrieval and code-tool trajectories to Qwen2.5 students; its self-consistent action generation also incurs candidate-generation and tool-execution costs [155]. These studies motivate evaluating complete interactions after compression.

History and communication changes can alter subsequent work. ACON learns compression guidelines from trajectories that fail after condensation, showing that lower peak context demand can coexist with extra compression calls, interaction steps or disrupted cache reuse [156]. AgentPrune reduces connections in a spatial-temporal communication graph, directly reducing messaging [157]. These application examples extend beyond the core model taxonomy and illustrate why peak size alone is an incomplete task-cost measure.

A suitable protocol fixes the environment, tools, and stopping rules, then pairs success with total tokens, model/tool calls, latency, and cost, including retries and failures. Testing model, history, reasoning-length, and communication interventions separately before combining them would reveal where information loss changes the later trajectory.

9.6. Limitations of This Review

This narrative synthesis prioritizes representative mechanisms and interpretable evidence for autoregressive text generation. Coverage depends on indexing, topic queries, citation tracing, and public reporting, which can favour successful configurations and omit proprietary or unpublished results. The selection flow summarizes author-reported screening totals; complete historical query logs and a study-level reconciliation of the 105 publications were not retained. The revision-stage numerical subset is purposively selected and does not estimate average effects across the field.

Results expressed relative to their own baselines still depend on source-specific calibration, recovery, tasks, and execution conditions. Small differences are often point estimates without reported variance. The S marker identifies separate quality and deployment experiments; P identifies paired measurements. The reporting appraisal in Section 2.5 therefore supports configuration-specific interpretation, with validation on the target workload needed for deployment.

Preparation, auxiliary storage, reconstruction, and complete-request costs are incompletely reported across studies. The quantitative deployment synthesis therefore centres on representation size, memory, latency, and throughput. Energy and monetary cost

lack sufficiently aligned device boundaries, utilization, electricity, or hardware prices and amortization horizons for a comparable synthesis. Evidence reflects the numerical checks through 13 October 2026 and the source-registry consolidation on 14 October 2026; rapid changes in kernels, hardware, and model families can change deployment outcomes. Generalization to multimodal and agent systems requires evaluation of their distinct information flows and success criteria.

10. Conclusions

This review connects compression mechanisms to retained capability and deployment choices for autoregressive text generation. Its source-relative synthesis distinguishes quality changes under compression, teacher-guided gains within compact students, and measured execution benefits. The method map identifies where to intervene; the numerical evidence and candidate checks show how reported results inform a target workload.

Three deployment principles emerge from the reviewed cases. Capacity constraints determine which representations enter a speed comparison. Within a feasible budget, task requirements can favour different allocations of precision, rank, or retained structure. Compact-model construction and request compression address complementary demands, connecting student capability to the history and generation needed for each task. Deployment selection therefore proceeds from the limiting resource, through required capability, to execution under the intended workload.

The next step is to test how these choices transfer across capabilities, changing contexts, and combined compression stages. Evaluation that includes delayed information use, preparation costs, and complete-task outcomes would support more reliable choices as models and workloads evolve.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/electronics15204826/s1>, Supplementary Evidence S1, containing numerical extraction, source-version records, and reporting appraisal for the quantitative synthesis.

Author Contributions: All authors conducted the literature research, wrote the original manuscript, and revised the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: This review reports no new model experiments. Supplementary Evidence S1 contains the current numerical extraction, source-version registry, and reporting appraisal in a separate spreadsheet. It preserves published baseline and compressed values, derived changes, and source locations; conversion rules are defined in Section 2.4.

Acknowledgments: ChatGPT and Codex (OpenAI, San Francisco, CA, USA; model: GPT-6.1 sol) assisted with organizing the literature reviewed in this manuscript, drafting manuscript sections, and revising their language. The authors manually checked the cited references and key numerical results against the source publications. The authors are responsible for the selection and interpretation of cited evidence and for the final manuscript.

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

References

1. Pope, R.; Douglas, S.; Chowdhery, A.; Devlin, J.; Bradbury, J.; Levskaya, A.; Heek, J.; Xiao, K.; Agrawal, S.; Dean, J. Efficiently Scaling Transformer Inference. In Proceedings of Machine Learning and Systems, Miami Beach, FL, USA, 4–8 June 2023; Volume 5, pp. 606–624. Available online: https://proceedings.mlsys.org/paper_files/paper/2023/hash/c4be71ab8d24cdfb45e3d06dbfca2780-Abstract-mlsys2023.html (accessed on 15 October 2026).

2. Liu, Z.; Yuan, J.; Jin, H.; Zhong, S.; Xu, Z.; Braverman, V.; Chen, B.; Hu, X. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *Proceedings of the 41st International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2024; Volume 235, pp. 32332–32344. Available online: <https://proceedings.mlr.press/v235/liu24bz.html> (accessed on 15 October 2026).
3. Zhu, X.; Li, J.; Liu, Y.; Ma, C.; Wang, W. A Survey on Model Compression for Large Language Models. *Trans. Assoc. Comput. Linguist.* **2024**, *12*, 1556–1577. https://doi.org/10.1162/tacl_a_00704.
4. Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; Chen, D. SnapKV: LLM Knows What You are Looking for Before Generation. In *Proceedings of the 38th Conference on Neural Information Processing Systems*, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 22947–22970. <https://doi.org/10.52202/079017-0722>.
5. Chang, C.C.; Lin, W.C.; Lin, C.Y.; Chen, C.Y.; Hu, Y.F.; Wang, P.S.; Huang, N.C.; Ceze, L.; Abdelfattah, M.S.; Wu, K.C. Palu: KV-Cache Compression with Low-Rank Projection. In *Proceedings of the International Conference on Learning Representations*, Singapore, 24–28 April 2025; Volume 2025, pp. 50222–50249. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/hash/7da6e0e00702c60607a6ae05c802ef85-Abstract-Conference.html (accessed on 15 October 2026).
6. 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*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 13358–13376. <https://doi.org/10.18653/v1/2023.emnlp-main.825>.
7. Ge, T.; Hu, J.; Wang, L.; Wang, X.; Chen, S.Q.; Wei, F. In-context Autoencoder for Context Compression in a Large Language Model. In *Proceedings of the International Conference on Learning Representations*, Vienna, Austria, 7–11 May 2024; pp. 2591–2607. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/hash/0b276510ec2d3f6613a8b60c41ff0438-Abstract-Conference.html (accessed on 15 October 2026).
8. Xia, H.; Leong, C.T.; Wang, W.; Li, Y.; Li, W. TokenSkip: Controllable Chain-of-Thought Compression in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 3351–3363. <https://doi.org/10.18653/v1/2025.emnlp-main.165>.
9. Wan, Z.; Wang, X.; Liu, C.; Alam, S.; Zheng, Y.; Liu, J.; Qu, Z.; Yan, S.; Zhu, Y.; Zhang, Q.; et al. Efficient Large Language Models: A Survey. *Trans. Mach. Learn. Res.* **2024**. Available online: <https://jmlr.org/tmlr/papers/bib/bsCCJHbO8A.bib> (accessed on 15 October 2026).
10. Zhou, Z.; Ning, X.; Hong, K.; Fu, T.; Xu, J.; Li, S.; Lou, Y.; Wang, L.; Yuan, Z.; Li, X.; et al. A Survey on Efficient Inference for Large Language Models. *arXiv* **2024**, arXiv:2404.14294v3. <https://doi.org/10.48550/arXiv.2404.14294>.
11. Li, H.; Li, Y.; Tian, A.; Tang, T.; Xu, Z.; Chen, X.; Hu, N.; Dong, W.; Li, Q.; Chen, L. A Survey on Large Language Model Acceleration based on KV Cache Management. *Trans. Mach. Learn. Res.* **2025**. Available online: <https://jmlr.org/tmlr/papers/bib/z3JZzu9EA3.bib> (accessed on 15 October 2026).
12. Chen, Y.D.; Zheng, K.J.; Guo, Z.H.; Zhang, Q.H.; Zhang, Y.H.; Zhai, J.D. A Survey of Quantization in LLM: Unlocking Potential Hardware Efficiency. *J. Comput. Sci. Technol.* **2026**, *41*, 341–358. <https://doi.org/10.1007/s11390-026-5979-1>.
13. Chen, W.; Wang, J.; Zhang, Y.; Shi, Y.; Jiang, T.; Zhou, S.; Wu, C.; Zhang, A.; Zhou, C.; Wang, M.; et al. On-device large language models: A survey of model compression and system optimization. *Artif. Intell. Rev.* **2026**, *59*, 191. <https://doi.org/10.1007/s10462-026-11538-1>.
14. Li, Z.; Liu, Y.; Su, Y.; Collier, N. Prompt Compression for Large Language Models: A Survey. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 7182–7195. <https://doi.org/10.18653/v1/2025.naacl-long.368>.
15. Lin, Y.; Tang, H.; Yang, S.; Zhang, Z.; Xiao, G.; Gan, C.; Han, S. QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving. In *Proceedings of the Machine Learning and Systems*, Santa Clara, CA, USA, 12–15 May 2025; Volume 7. Available online: https://proceedings.mlsys.org/paper_files/paper/2025/hash/fbe2b2f74a2ece8070d8fb073717bda6-Abstract-Conference.html (accessed on 15 October 2026).
16. Chen, A.; Geh, R.; Grover, A.; Van Den Broeck, G.; Israel, D.M. The Pitfalls of KV Cache Compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2026; pp. 41530–41553. <https://doi.org/10.18653/v1/2026.acl-long.1926>.
17. Dong, P.; Tang, Z.; Liu, X.; Li, L.; Chu, X.; Li, B. Can Compressed LLMs Truly Act? An Empirical Evaluation of Agentic Capabilities in LLM Compression. In *Proceedings of the 42nd International Conference on Machine Learning*, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 14169–14202. <https://proceedings.mlr.press/v267/dong25k.html>.
18. Łajewska, W.; Hardalov, M.; Aina, L.; Anna John, N.; Su, H.; Marquez, L. Understanding and Improving Information Preservation in Prompt Compression for LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 17520–17541. <https://doi.org/10.18653/v1/2025.findings-emnlp.949>.
19. Deng, C.; Zhang, Z.; Mao, K.; Li, S.; Huang, X.; Yu, D.; Dou, Z. A Silver Bullet or a Compromise for Full Attention? A Comprehensive Study of Gist Token-based Context Compression. In *Proceedings of the 63rd Annual Meeting of the Association*

- for *Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 4861–4879. <https://doi.org/10.18653/v1/2025.acl-long.241>.
20. Page, M.J.; McKenzie, J.E.; Bossuyt, P.M.; Boutron, I.; Hoffmann, T.C.; Mulrow, C.D.; Shamseer, L.; Tetzlaff, J.M.; Akl, E.A.; Brennan, S.E.; et al. The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ* **2021**, *372*, n71. <https://doi.org/10.1136/bmj.n71>.
 21. Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C.H.; Gonzalez, J.; Zhang, H.; Stoica, I. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*; Association for Computing Machinery: New York, NY, USA, 2023; pp. 611–626. <https://doi.org/10.1145/3600006.3613165>.
 22. Dao, T. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *Proceedings of the International Conference on Learning Representations*, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 35549–35562. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/hash/98ed250b203d1ac6b24bbcf263e3d4a7-Abstract-Conference.html (accessed on 15 October 2026).
 23. Sheng, Y.; Zheng, L.; Yuan, B.; Li, Z.; Ryabinin, M.; Chen, B.; Liang, P.; Re, C.; Stoica, I.; Zhang, C. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, Honolulu, Hawaii, USA, 23–29 July 2023; Volume 202, pp. 31094–31116. Available online: <https://proceedings.mlr.press/v202/sheng23a.html> (accessed on 15 October 2026).
 24. Ko, J.; Kim, S.; Chen, T.; Yun, S.Y. DistiLLM: Towards Streamlined Distillation for Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 24872–24895. <https://proceedings.mlr.press/v235/ko24c.html>.
 25. Saha, R.; Sagan, N.; Srivastava, V.; Goldsmith, A.J.; Pilanci, M. Compressing Large Language Models using Low Rank and Low Precision Decomposition. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems*, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 88981–89018. <https://doi.org/10.52202/079017-2823>.
 26. Frantar, E.; Ashkboos, S.; Hoefler, T.; Alistarh, D. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *Proceedings of The Eleventh International Conference on Learning Representations*, Kigali, Rwanda, 1–5 May 2023. Available online: <https://iclr.cc/virtual/2023/poster/10855> (accessed on 15 October 2026).
 27. Liu, S.; He, B.; Wu, H.; Song, L. Beyond One-Size-Fits-All Pruning via Evolutionary Metric Search for Large Language Models. *arXiv* **2025**, arXiv:2502.10735. <https://doi.org/10.48550/arXiv.2502.10735>.
 28. Tseng, A.; Chee, J.; Sun, Q.; Kuleshov, V.; De Sa, C. QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 48630–48656. Available online: <https://proceedings.mlr.press/v235/tseng24a.html> (accessed on 15 October 2026).
 29. Li, G.; Xu, Y.; Li, Z.; Liu, J.; Yin, X.; Li, D.; Barsoum, E. Týr-the-Pruner: Structural Pruning LLMs via Global Sparsity Distribution Optimization. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025)*, San Diego, CA, USA, 2–7 December 2025; Volume 38, pp. 149163–149192. Available online: https://proceedings.neurips.cc/paper_files/paper/2025/hash/db2cbf43a349bc866111e791b58c7bf4-Abstract-Conference.html (accessed on 15 October 2026). <https://doi.org/10.52202/085713-4989>.
 30. Xiao, G.; Lin, J.; Seznec, M.; Wu, H.; Demouth, J.; Han, S. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, Honolulu, HI, USA, 23–29 July 2023; Volume 202, pp. 38087–38099. Available online: <https://proceedings.mlr.press/v202/xiao23c.html> (accessed on 15 October 2026).
 31. Kim, S.; Hooper, C.R.C.; Gholami, A.; Dong, Z.; Li, X.; Shen, S.; Mahoney, M.W.; Keutzer, K. SqueezeLLM: Dense-and-Sparse Quantization. In *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 23901–23923. Available online: <https://proceedings.mlr.press/v235/kim24f.html> (accessed on 15 October 2026).
 32. Egiazarian, V.; Panferov, A.; Kuznedelev, D.; Frantar, E.; Babenko, A.; Alistarh, D. Extreme Compression of Large Language Models via Additive Quantization. In *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 12284–12303. Available online: <https://proceedings.mlr.press/v235/egiazarian24a.html> (accessed on 15 October 2026).
 33. Shao, W.; Chen, M.; Zhang, Z.; Xu, P.; Zhao, L.; Li, Z.; Zhang, K.; Gao, P.; Qiao, Y.; Luo, P. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. In *Proceedings of the International Conference on Learning Representations*, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 45472–45496. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/hash/c6483c8a68083af3383f91ee0dc6db95-Abstract-Conference.html (accessed on 15 October 2026).
 34. Liu, Z.; Oguz, B.; Zhao, C.; Chang, E.; Stock, P.; Mehdad, Y.; Shi, Y.; Krishnamoorthi, R.; Chandra, V. LLM-QAT: Data-Free Quantization Aware Training for Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024, pp. 467–484. <https://doi.org/10.18653/v1/2024.findings-acl.26>.

35. Chen, M.; Shao, W.; Xu, P.; Wang, J.; Gao, P.; Zhang, K.; Luo, P. EfficientQAT: Efficient Quantization-Aware Training for Large Language Models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025, pp. 10081–10100. <https://doi.org/10.18653/v1/2025.acl-long.498>.
36. 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 On-Device LLM Compression and Acceleration. In *Proceedings of the Machine Learning and Systems*, Santa Clara, CA, USA, 13–16 May 2024; Volume 6, pp. 87–100. Available online: https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html (accessed on 15 October 2026).
37. Ashkboos, S.; Mohtashami, A.; Croci, M.L.; Li, B.; Cameron, P.; Jaggi, M.; Alistarh, D.; Hoefler, T.; Hensman, J. QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs. In *Proceedings of the Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 100213–100240. <https://doi.org/10.52202/079017-3180>.
38. Liu, Z.; Zhao, C.; Fedorov, I.; Soran, B.; Choudhary, D.; Krishnamoorthi, R.; Chandra, V.; Tian, Y.; Blankevoort, T. SpinQuant: LLM Quantization with Learned Rotations. In *Proceedings of the International Conference on Learning Representations*, Singapore, 24–28 April 2025; Volume 2025, pp. 92009–92032. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/hash/e5b1c0d4866f72393c522c8a00eed4eb-Abstract-Conference.html (accessed on 15 October 2026).
39. Sun, Y.; Liu, R.; Bai, H.; Bao, H.; Zhao, K.; Li, Y.; Hu, J.; Yu, X.; Hou, L.; Yuan, C.; et al. FlatQuant: Flatness Matters for LLM Quantization. In *Proceedings of the 42nd International Conference on Machine Learning*, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 57587–57613. Available online: <https://proceedings.mlr.press/v267/sun251.html> (accessed on 15 October 2026).
40. Ashkboos, S.; Markov, I.; Frantar, E.; Zhong, T.; Wang, X.; Ren, J.; Hoefler, T.; Alistarh, D. QUIK: Towards End-to-end 4-Bit Inference on Generative Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 3355–3371. <https://doi.org/10.18653/v1/2024.emnlp-main.197>.
41. Zhao, Y.; Lin, C.Y.; Zhu, K.; Ye, Z.; Chen, L.; Zheng, S.; Ceze, L.; Krishnamurthy, A.; Chen, T.; Kasikci, B. Atom: Low-Bit Quantization for Efficient and Accurate LLM Serving. In *Proceedings of Machine Learning and Systems*, Santa Clara, CA, USA, 13–16 May 2024; Volume 6, pp. 196–209. Available online: https://proceedings.mlsys.org/paper_files/paper/2024/hash/5edb57c05c81d04beb716ef1d542fe9e-Abstract-Conference.html (accessed on 15 October 2026).
42. Tseng, A.; Sun, Q.; Hou, D.; De Sa, C. QTIP: Quantization with Trellises and Incoherence Processing. In *Proceedings of the Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 59597–59620. <https://doi.org/10.52202/079017-1904>.
43. Hooper, C.; Kim, S.; Mohammadzadeh, H.; Mahoney, M.W.; Shao, Y.S.; Keutzer, K.; Gholami, A. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. In *Proceedings of the Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 1270–1303. <https://doi.org/10.52202/079017-0040>.
44. Duanmu, H.; Yuan, Z.; Li, X.; Duan, J.; Zhang, X.; Lin, D. SKVQ: Sliding-window Key and Value Cache Quantization for Large Language Models. In *Proceedings of the First Conference on Language Modeling*, Philadelphia, PA, USA, 7–9 October 2024. Available online: <https://arxiv.org/abs/2405.06219> (accessed on 15 October 2026).
45. Sharma, A.; Ding, H.; Li, J.; Dani, N.; Zhang, M. MiniKV: Pushing the Limits of 2-Bit KV Cache via Compression and System Co-Design for Efficient Long Context Inference. In *Findings of the Association for Computational Linguistics: ACL 2025*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 18506–18523. <https://doi.org/10.18653/v1/2025.findings-acl.952>.
46. Guo, H.; Brandon, W.; Cholakov, R.; Ragan-Kelley, J.; Xing, E.P.; Kim, Y. Fast Matrix Multiplications for Lookup Table-Quantized LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 12419–12433. <https://doi.org/10.18653/v1/2024.findings-emnlp.724>.
47. Park, G.; Park, B.; Kim, M.; Lee, S.; Kim, J.; Kwon, B.; Kwon, S.J.; Kim, B.; Lee, Y.; Lee, D. LUT-GEMM: Quantized Matrix Multiplication based on LUTs for Efficient Inference in Large-Scale Generative Language Models. In *Proceedings of the International Conference on Learning Representations*, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 38069–38086. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/hash/a4f98ce85f440ee269b0df57b4368719-Abstract-Conference.html (accessed on 15 October 2026).
48. Wei, J.; Cao, S.; Cao, T.; Ma, L.; Wang, L.; Zhang, Y.; Yang, M. T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge. In *Proceedings of the Twentieth European Conference on Computer Systems*, Rotterdam, The Netherlands, 30 March–3 April 2025; pp. 278–292. <https://doi.org/10.1145/3689031.3696099>.
49. Xia, H.; Zheng, Z.; Wu, X.; Chen, S.; Yao, Z.; Youn, S.; Bakhtiari, A.; Wyatt, M.; Zhuang, D.; Zhou, Z.; et al. Quant-LLM: Accelerating the Serving of Large Language Models via FP6-Centric Algorithm-System Co-Design on Modern GPUs. In *Proceedings of the 2024 USENIX Annual Technical Conference (USENIX ATC 24)*, Santa Clara, CA, USA, 10–12 July 2024; pp. 699–713. Available online: <https://www.usenix.org/conference/atc24/presentation/xia> (accessed on 15 October 2026).

50. Gong, R.; Yong, Y.; Gu, S.; Huang, Y.; Lv, C.; Zhang, Y.; Tao, D.; Liu, X. LLMC: Benchmarking Large Language Model Quantization with a Versatile Compression Toolkit. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 132–152. <https://doi.org/10.18653/v1/2024.emnlp-industry.12>.
51. Lee, S.; Woo, S.T.; Jin, J.; Lee, C.; Park, E. AMQ: Enabling AutoML for Mixed-precision Weight-Only Quantization of Large Language Models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 35532–35550. <https://doi.org/10.18653/v1/2025.emnlp-main.1799>.
52. Wei, X.; Zhang, Y.; Li, Y.; Zhang, X.; Gong, R.; Guo, J.; Liu, X. Outlier Suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 1648–1665. <https://doi.org/10.18653/v1/2023.emnlp-main.102>.
53. Ma, Y.; Li, H.; Zheng, X.; Ling, F.; Xiao, X.; Wang, R.; Wen, S.; Chao, F.; Ji, R. AffineQuant: Affine Transformation Quantization for Large Language Models. In *Proceedings of the International Conference on Learning Representations*, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 50932–50951. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/hash/ddfb58b61e7213dfb9e06c695475b2bd-Abstract-Conference.html (accessed on 15 October 2026).
54. Liu, J.; Gong, R.; Wei, X.; Dong, Z.; Cai, J.; Zhuang, B. QLLM: Accurate and Efficient Low-Bitwidth Quantization for Large Language Models. In *Proceedings of the International Conference on Learning Representations*, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 15417–15439. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/hash/437bc4ccaf3fc6d4289bd10940be42b-Abstract-Conference.html (accessed on 15 October 2026).
55. Lin, H.; Xu, H.; Wu, Y.; Cui, J.; Zhang, Y.; Mou, L.; Song, L.; Sun, Z.; Wei, Y. DuQuant: Distributing Outliers via Dual Transformation Makes Stronger Quantized LLMs. In *Proceedings of the Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 87766–87800. Available online: https://proceedings.neurips.cc/paper_files/paper/2024/hash/9febda1c8344cc5f2d51713964864e93-Abstract-Conference.html (accessed on 15 October 2026). <https://doi.org/10.52202/079017-2786>.
56. Hu, X.; Cheng, Y.; Yang, D.; Xu, Z.; Zhou, S.; Yu, J.; Xu, C.; Chen, Z.; Jiang, Z.; Yuan, Z. OSTQuant: Refining Large Language Model Quantization with Orthogonal and Scaling Transformations for Better Distribution Fitting. In *Proceedings of the International Conference on Learning Representations*, Singapore, 24–28 April 2025; Volume 2025, pp. 37492–37517. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/hash/5cebc89b113920dbff7c79854ba765a3-Abstract-Conference.html (accessed on 15 October 2026).
57. van Breugel, B.; Bondarenko, Y.; Whatmough, P.N.; Nagel, M. FPTQuant: Function-Preserving Transforms for LLM Quantization. In *Proceedings of the 43rd International Conference on Machine Learning*, Seoul, Republic of Korea, 6–11 July 2026; Volume 306, pp. 123355–123382. Available online: <https://proceedings.mlr.press/v306/van-breugel26a.html> (accessed on 15 October 2026).
58. 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*, Honolulu, HI, USA, 23–29 July 2023; Volume 202, pp. 10323–10337. Available online: <https://proceedings.mlr.press/v202/frantar23a.html> (accessed on 15 October 2026).
59. Ma, X.; Fang, G.; Wang, X. LLM-Pruner: On the Structural Pruning of Large Language Models. In *Proceedings of the Advances in Neural Information Processing Systems*, New Orleans, LA, USA, 10–16 December 2023; Volume 36, pp. 21702–21720. <https://doi.org/10.52202/075280-0950>.
60. Fang, G.; Yin, H.; Muralidharan, S.; Heinrich, G.; Pool, J.; Kautz, J.; Molchanov, P.; Wang, X. MaskLLM: Learnable Semi-Structured Sparsity for Large Language Models. In *Proceedings of the Advances in Neural Information Processing Systems*, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 7736–7758. <https://doi.org/10.52202/079017-0248>.
61. Liu, Z.; Wang, J.; Dao, T.; Zhou, T.; Yuan, B.; Song, Z.; Shrivastava, A.; Zhang, C.; Tian, Y.; Ré, C.; et al. Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time. In *Proceedings of the 40th International Conference on Machine Learning*, Honolulu, HI, USA, 23–29 July 2023; Volume 202, pp. 22137–22176. Available online: <https://proceedings.mlr.press/v202/liu23am.html> (accessed on 15 October 2026).
62. Dong, H.; Chen, B.; Chi, Y. Prompt-prompted Adaptive Structured Pruning for Efficient LLM Generation. In *Proceedings of the First Conference on Language Modeling*, Philadelphia, PA, USA, 7–9 October 2024. Available online: <https://arxiv.org/abs/2404.01365> (accessed on 15 October 2026).
63. Le, Q.; Diao, E.; Wang, Z.; Wang, X.; Ding, J.; Yang, L.; Anwar, A. Probe Pruning: Accelerating LLMs through Dynamic Pruning via Model-Probing. In *Proceedings of the International Conference on Learning Representations*, Singapore, 24–28 April 2025; Volume 2025, pp. 72494–72513. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/b44efaa3b7a389ea56c7ee9d6589e800-Paper-Conference.pdf (accessed on 15 October 2026).
64. 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*, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 4942–4964. Available

- online: https://proceedings.iclr.cc/paper_files/paper/2024/file/14c856c7a41297804de4c4890e846b25-Paper-Conference.pdf (accessed on 15 October 2026).
65. Yin, L.; Wu, Y.; Zhang, Z.; Hsieh, C.Y.; Wang, Y.; Jia, Y.; Li, G.; Jaiswal, A.K.; Pechenizkiy, M.; Liang, Y.; et al. Outlier Weighed Layerwise Sparsity (OWL): A Missing Secret Sauce for Pruning LLMs to High Sparsity. In Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 57101–57115. Available online: <https://proceedings.mlr.press/v235/yin24e.html> (accessed on 15 October 2026).
 66. Meng, X.; Behdin, K.; Wang, H.; Mazumder, R. ALPS: Improved Optimization for Highly Sparse One-Shot Pruning for Large Language Models. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 37594–37625. <https://doi.org/10.52202/079017-1187>.
 67. Zhang, Y.; Bai, H.; Lin, H.; Zhao, J.; Hou, L.; Cannistraci, C.V. Plug-and-Play: An Efficient Post-training Pruning Method for Large Language Models. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 50490–50508. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/db6ccc979860d7a233ecaf588bb23512-Paper-Conference.pdf (accessed on 15 October 2026).
 68. Ashkboos, S.; Croci, M.L.; Gennari do Nascimento, M.; Hoefler, T.; Hensman, J. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 11682–11701. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/316648eb8b4ff6010f531b07848c300-Paper-Conference.pdf (accessed on 15 October 2026).
 69. An, Y.; Zhao, X.; Yu, T.; Tang, M.; Wang, J. Fluctuation-Based Adaptive Structured Pruning for Large Language Models. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 20–27 February 2024; Volume 38, No. 10, pp. 10865–10873. <https://doi.org/10.1609/aaai.v38i10.28960>.
 70. Men, X.; Xu, M.; Zhang, Q.; Yuan, Q.; Wang, B.; Lin, H.; Lu, Y.; Han, X.; Chen, W. ShortGPT: Layers in Large Language Models are More Redundant Than You Expect. In *Findings of the Association for Computational Linguistics: ACL 2025*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 20192–20204. <https://doi.org/10.18653/v1/2025.findings-acl.1035>.
 71. Xia, M.; Gao, T.; Zeng, Z.; Chen, D. Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 5385–5409. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/160adf2dc118a920e7858484b92a37d8-Paper-Conference.pdf (accessed on 15 October 2026).
 72. Liu, J.; Ponnusamy, P.; Cai, T.; Guo, H.; Kim, Y.; Athiwaratkun, B. Training-Free Activation Sparsity in Large Language Models. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 98302–98322. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/f3f2ff9579ba6deeb89caa2fe1f0b99c-Paper-Conference.pdf (accessed on 15 October 2026).
 73. Lee, D.; Lee, J.Y.; Zhang, G.; Tiwari, M.; Mirhoseini, A. CATS: Contextually-Aware Thresholding for Sparsity in Large Language Models. In Proceedings of the First Conference on Language Modeling, Philadelphia, PA, USA, 7–9 October 2024. Available online: <https://arxiv.org/abs/2404.08763> (accessed on 15 October 2026).
 74. Song, C.; Han, X.; Zhang, Z.; Hu, S.; Shi, X.; Li, K.; Chen, C.; Liu, Z.; Li, G.; Yang, T.; et al. ProSparse: Introducing and Enhancing Intrinsic Activation Sparsity within Large Language Models. In Proceedings of the 31st International Conference on Computational Linguistics; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 2626–2644. Available online: <https://aclanthology.org/2025.coling-main.180/> (accessed on 15 October 2026).
 75. Ji, Y.; Xiang, Y.; Li, J.; Xia, Q.; Li, P.; Duan, X.; Wang, Z.; Zhang, M. Beware of Calibration Data for Pruning Large Language Models. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 18604–18620. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/2ede933e10afa991a10b6f36b6522129-Paper-Conference.pdf (accessed on 15 October 2026).
 76. Zhang, Y.; Zhao, L.; Lin, M.; Sun, Y.; Yao, Y.; Han, X.; Tanner, J.; Liu, S.; Ji, R. Dynamic Sparse No Training: Training-Free Fine-tuning for Sparse LLMs. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 249–264. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/0147d967a5db3b8dde08d2a327b24568-Paper-Conference.pdf (accessed on 15 October 2026).
 77. Liu, Y.; Wu, F.; Pan, X.; Li, R.; Tang, Z.; Li, K. LaCo: Layer-wise Compensation for Pruned Large Language Models. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), San Diego, CA, USA, 2–7 July 2026; Association for Computational Linguistics: Stroudsburg, PA, USA, 2026; pp. 29099–29113. <https://doi.org/10.18653/v1/2026.acl-long.1342>.
 78. Zhang, M.; Chen, H.; Shen, C.; Yang, Z.; Ou, L.; Yu, X.; Zhuang, B. LoRAPrune: Structured Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 3013–3026. <https://doi.org/10.18653/v1/2024.findings-acl.178>.
 79. Liu, P.; Zeng, H.; Zhang, Y.; Yi, Y.; Wu, J. Resource-Efficient Pruning for Transformer via Low-Rank Importance Estimation. *IEEE Trans. Emerg. Top. Comput. Intell.* **2026**, 1–15. <https://doi.org/10.1109/TETCI.2026.3727362>.

80. Muralidharan, S.; Sreenivas, S.T.; Joshi, R.; Chochowski, M.; Patwary, M.; Shoeybi, M.; Catanzaro, B.; Kautz, J.; Molchanov, P. Compact Language Models via Pruning and Knowledge Distillation. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 41076–41102. <https://doi.org/10.52202/079017-1299>.
81. Zhang, S.; Pappayan, V. OATS: Outlier-Aware Pruning Through Sparse and Low Rank Decomposition. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 50870–50895. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/7f2fc4053a66edfa430bcd9a6ff3b17-Paper-Conference.pdf (accessed on 15 October 2026).
82. Dong, P.; Li, L.; Zhong, Y.; Du, D.; Fan, R.; Chen, Y.; Tang, Z.; Wang, Q.; Xue, W.; Guo, Y.; et al. STBLLM: Breaking the 1-Bit Barrier with Structured Binary LLMs. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 102882–102907. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/ff997469ac66cf893c4183efeb22212a-Paper-Conference.pdf (accessed on 15 October 2026).
83. Xia, H.; Zheng, Z.; Li, Y.; Zhuang, D.; Zhou, Z.; Qiu, X.; Li, Y.; Lin, W.; Song, S.L. Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity. *Proc. VLDB Endow.* **2023**, *17*, 211–224. <https://doi.org/10.14778/3626292.3626303>.
84. Song, Y.; Mi, Z.; Xie, H.; Chen, H. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. In Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, Austin, TX, USA, 4–6 November 2024; pp. 590–606. <https://doi.org/10.1145/3694715.3695964>.
85. Kim, J.; Lee, Y.; Do, J. Unified Static-Dynamic Pruning for Efficient LLM Inference. *Proc. VLDB Endow.* **2026**, *19*, 2950–2963. <https://doi.org/10.14778/3836663.3836665>.
86. Dong, P.; Li, L.; Tang, Z.; Liu, X.; Pan, X.; Wang, Q.; Chu, X. Pruner-Zero: Evolving Symbolic Pruning Metric From Scratch for Large Language Models. In Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 11346–11374. Available online: <https://proceedings.mlr.press/v235/dong24b.html> (accessed on 15 October 2026).
87. Yang, Y.; Cao, Z.; Zhao, H. LaCo: Large Language Model Pruning via Layer Collapse. In *Findings of the Association for Computational Linguistics: EMNLP 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 6401–6417. <https://doi.org/10.18653/v1/2024.findings-emnlp.372>.
88. Kurtic, E.; Frantar, E.; Alistarh, D. ZipLM: Inference-Aware Structured Pruning of Language Models. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December 2023; Volume 36, pp. 65597–65617. <https://doi.org/10.52202/075280-2863>.
89. Meng, X.; Ibrahim, S.; Behdin, K.; Hazimeh, H.; Ponomareva, N.; Mazumder, R. OSSCAR: One-Shot Structured Pruning in Vision and Language Models with Combinatorial Optimization. In Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 35354–35377. Available online: <https://proceedings.mlr.press/v235/meng24a.html> (accessed on 15 October 2026).
90. Huang, W.; Zhang, Y.; Zheng, X.; Chao, F.; Ji, R. Determining Layer-wise Sparsity for Large Language Models Through a Theoretical Perspective. In Proceedings of the 42nd International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 26175–26201. Available online: <https://proceedings.mlr.press/v267/huang25ax.html> (accessed on 15 October 2026).
91. Lee, D.; Lee, K.; Chung, J.; Lee, N. SAFE: Finding Sparse and Flat Minima to Improve Pruning. In Proceedings of the 42nd International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 33300–33321. Available online: <https://proceedings.mlr.press/v267/lee25s.html> (accessed on 15 October 2026).
92. Kim, Y.; Rush, A.M. Sequence-Level Knowledge Distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2016; pp. 1317–1327. <https://doi.org/10.18653/v1/D16-1139>.
93. Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; Liu, Q. TinyBERT: Distilling BERT for Natural Language Understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 4163–4174. <https://doi.org/10.18653/v1/2020.findings-emnlp.372>.
94. Wang, W.; Wei, F.; Dong, L.; Bao, H.; Yang, N.; Zhou, M. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. In Proceedings of the Advances in Neural Information Processing Systems, Virtual Conference, 6–12 December 2020; Volume 33, pp. 5776–5788. Available online: <https://proceedings.neurips.cc/paper/2020/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html> (accessed on 15 October 2026).
95. Gu, Y.; Dong, L.; Wei, F.; Huang, M. MiniLLM: Knowledge Distillation of Large Language Models. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 32694–32717. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/8ac015d409635f196f9e3e9dcfb9a94e-Paper-Conference.pdf (accessed on 15 October 2026).

96. Agarwal, R.; Vieillard, N.; Zhou, Y.; Stanczyk, P.; Ramos Garea, S.; Geist, M.; Bachem, O. On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 21246–21263. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/5be69a584901a26c521c2b51e40a4c20-Paper-Conference.pdf (accessed on 15 October 2026).
97. Ko, J.; Chen, T.; Kim, S.; Ding, T.; Liang, L.; Zharkov, I.; Yun, S.Y. DistiLLM-2: A Contrastive Approach Boosts the Distillation of LLMs. In Proceedings of the 42nd International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 31044–31062. Available online: <https://proceedings.mlr.press/v267/ko25a.html> (accessed on 15 October 2026).
98. Shing, M.; Misaki, K.; Bao, H.; Yokoi, S.; Akiba, T. TAID: Temporally Adaptive Interpolated Distillation for Efficient Knowledge Transfer in Language Models. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 92233–92256. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/e664650506f1cf2b4696df892147c06e-Paper-Conference.pdf (accessed on 15 October 2026).
99. Kim, Y.; Shin, D.; Kang, M.; Na, B.; Moon, I.C. Distillation of Large Language Models via Concrete Score Matching. In Proceedings of the International Conference on Learning Representations, Rio de Janeiro, Brazil, 23–27 April 2026; Volume 2026, pp. 16649–16682. Available online: https://proceedings.iclr.cc/paper_files/paper/2026/file/1bfc9f74afa91b9b8add5a97a97001a1-Paper-Conference.pdf (accessed on 15 October 2026).
100. Xu, W.; Han, R.; Wang, Z.; Le, L.T.; Madeka, D.; Li, L.; Wang, W.Y.; Agarwal, R.; Lee, C.Y.; Pfister, T. Speculative Knowledge Distillation: Bridging the Teacher-Student Gap Through Interleaved Sampling. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 64616–64646. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/a2747a3844ca1e4667fbff3f558eb39b-Paper-Conference.pdf (accessed on 15 October 2026).
101. Gu, Y.; Zhou, H.; Meng, F.; Zhou, J.; Huang, M. MiniPLM: Knowledge Distillation for Pre-training Language Models. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 93661–93687. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/ea05e4fc0299c27648c9985266abad47-Paper-Conference.pdf (accessed on 15 October 2026).
102. Honovich, O.; Scialom, T.; Levy, O.; Schick, T. Unnatural Instructions: Tuning Language Models with (Almost) No Human Labor. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, ON, Canada, 9–14 July 2023; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 14409–14428. <https://doi.org/10.18653/v1/2023.acl-long.806>.
103. Xu, C.; Sun, Q.; Zheng, K.; Geng, X.; Zhao, P.; Feng, J.; Tao, C.; Lin, Q.; Jiang, D. WizardLM: Empowering Large Pre-Trained Language Models to Follow Complex Instructions. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 30745–30766. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/82eec786fd6bfa53450c5feb7d1ac92-Paper-Conference.pdf (accessed on 15 October 2026).
104. Zhou, Y.; Zhu, J.; Xu, P.; Liu, X.; Wang, X.; Koutra, D.; Ai, W.; Huang, F. Multi-Stage Balanced Distillation: Addressing Long-Tail Challenges in Sequence-Level Knowledge Distillation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 3315–3333. <https://doi.org/10.18653/v1/2024.findings-emnlp.190>.
105. Zhou, Y.; Lyu, K.; Rawat, A.S.; Menon, A.K.; Rostamizadeh, A.; Kumar, S.; Kagy, J.F.; Agarwal, R. DistillSpec: Improving Speculative Decoding via Knowledge Distillation. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 32011–32050. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/8766fbc68e1ed1cdef712ce273e0a363-Paper-Conference.pdf (accessed on 15 October 2026).
106. Ho, N.; Schmid, L.; Yun, S.Y. Large Language Models Are Reasoning Teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 14852–14882. <https://doi.org/10.18653/v1/2023.acl-long.830>.
107. Li, L.H.; Hessel, J.; Yu, Y.; Ren, X.; Chang, K.W.; Choi, Y. Symbolic Chain-of-Thought Distillation: Small Models Can Also “Think” Step-by-Step. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 2665–2679. <https://doi.org/10.18653/v1/2023.acl-long.150>.
108. Hsieh, C.Y.; Li, C.L.; Yeh, C.K.; Nakhost, H.; Fujii, Y.; Ratner, A.; Krishna, R.; Lee, C.Y.; Pfister, T. Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 8003–8017. <https://doi.org/10.18653/v1/2023.findings-acl.507>.
109. Shridhar, K.; Stolfo, A.; Sachan, M. Distilling Reasoning Capabilities into Smaller Language Models. In *Findings of the Association for Computational Linguistics: ACL 2023*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 7059–7073. <https://doi.org/10.18653/v1/2023.findings-acl.441>.

110. Wang, P.; Wang, Z.; Li, Z.; Gao, Y.; Yin, B.; Ren, X. SCOTT: Self-Consistent Chain-of-Thought Distillation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 5546–5558. <https://doi.org/10.18653/v1/2023.acl-long.304>.
111. 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>.
112. Chen, X.; Zhou, S.; Liang, K.; Sun, X.; Liu, X. Skip-Thinking: Chunk-wise Chain-of-Thought Distillation Enable Smaller Language Models to Reason Better and Faster. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 12142–12157. <https://doi.org/10.18653/v1/2025.emnlp-main.610>.
113. Ma, X.; Wan, G.; Yu, R.; Fang, G.; Wang, X. CoT-Valve: Length-Compressible Chain-of-Thought Tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 6025–6035. <https://doi.org/10.18653/v1/2025.acl-long.300>.
114. Han, T.; Wang, Z.; Fang, C.; Zhao, S.; Ma, S.; Chen, Z. Token-Budget-Aware LLM Reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 24842–24855. <https://doi.org/10.18653/v1/2025.findings-acl.1274>.
115. Boizard, N.; El Haddad, K.; Hudelot, C.; Colombo, P. Towards Cross-Tokenizer Distillation: The Universal Logit Distillation Loss for LLMs. *Trans. Mach. Learn. Res.* **2025**. Available online: <https://jmlr.org/tmlr/papers/bib/bwRxXiG09A.bib> (accessed on 15 October 2026).
116. Zhang, S.; Zhang, X.; Sun, Z.; Chen, Y.; Xu, J. Dual-Space Knowledge Distillation for Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 18164–18181. <https://doi.org/10.18653/v1/2024.emnlp-main.1010>.
117. Wang, J.; Paliotta, D.; May, A.; Rush, A.M.; Dao, T. The Mamba in the Llama: Distilling and Accelerating Hybrid Models. In *Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 10–15 December 2024*; Volume 37, pp. 62432–62457. <https://doi.org/10.52202/079017-1996>.
118. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531. <https://doi.org/10.48550/arXiv.1503.02531>.
119. Lin, A.; Wohlwend, J.; Chen, H.; Lei, T. Autoregressive Knowledge Distillation through Imitation Learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 6121–6133. <https://doi.org/10.18653/v1/2020.emnlp-main.494>.
120. Wang, Q.; Ke, J.; Tomizuka, M.; Keutzer, K.; Xu, C. Dobi-SVD: Differentiable SVD for LLM Compression and Some New Perspectives. In *Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025*; Volume 2025, pp. 12561–12590. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/218ca0d92e6ed8f9db00621e103dc70c-Paper-Conference.pdf (accessed on 15 October 2026).
121. Hsu, Y.C.; Hua, T.; Chang, S.E.; Lou, Q.; Shen, Y.; Jin, H. Language model compression with weighted low-rank factorization. In *Proceedings of the International Conference on Learning Representations, Virtual, 25–29 April 2022*. Available online: <https://arxiv.org/abs/2207.00112> (accessed on 15 October 2026).
122. Yuan, Z.; Shang, Y.; Song, Y.; Yang, D.; Wu, Q.; Yan, Y.; Sun, G. ASVD: Activation-aware Singular Value Decomposition for Compressing Large Language Models. *arXiv* **2023**, arXiv:2312.05821. <https://doi.org/10.48550/arXiv.2312.05821>.
123. Wang, X.; Zheng, Y.; Wan, Z.; Zhang, M. SVD-LLM: Truncation-aware Singular Value Decomposition for Large Language Model Compression. In *Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025*; Volume 2025, pp. 19299–19319. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/3104e1ab39875cf54fe1eb4473e7c5a1-Paper-Conference.pdf (accessed on 15 October 2026).
124. Wang, X.; Alam, S.; Wan, Z.; Shen, H.; Zhang, M. SVD-LLM V2: Optimizing Singular Value Truncation for Large Language Model Compression. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 4287–4296. <https://doi.org/10.18653/v1/2025.naacl-long.217>.
125. Sundrani, S.; Tudisco, F.; Minervini, P. Low-Rank Compression of Language Models via Differentiable Rank Selection. In *Proceedings of the Fifteenth Language Resources and Evaluation Conference*; ELRA Language Resources Association: Paris, France, 2026; pp. 10031–10045. <https://doi.org/10.63317/2xbs948bhby9>.
126. Lin, C.H.; Gao, S.; Smith, J.; Patel, A.; Tuli, S.; Shen, Y.; Jin, H.; Hsu, Y.C. MoDeGPT: Modular Decomposition for Large Language Model Compression. In *Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025*; Volume 2025, pp. 101355–101390. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/fb7214d2fd484165b08539d59c92e07-Paper-Conference.pdf (accessed on 15 October 2026).
127. Wang, J.; Chen, Y.G.; Lin, I.C.; Li, B.; Zhang, G.L. Basis Sharing: Cross-Layer Parameter Sharing for Large Language Model Compression. In *Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025*; Volume

- 2025, pp. 13386–13404. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/file/238c98450b1d9e8055f94d22f303bb57-Paper-Conference.pdf (accessed on 15 October 2026).
128. Li, W.; Li, L.; Gu, H.; Huang, Y.L.; Lee, M.G.; Sun, S.; Xue, W.; Guo, Y. MoE-SVD: Structured Mixture-of-Experts LLMs Compression via Singular Value Decomposition. In Proceedings of the 42nd International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 35209–35230. Available online: <https://proceedings.mlr.press/v267/li25az.html> (accessed on 15 October 2026).
 129. Li, Y.; Dong, B.; Guerin, F.; Lin, C. Compressing Context to Enhance Inference Efficiency of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 6342–6353. <https://doi.org/10.18653/v1/2023.emnlp-main.391>.
 130. Jiang, H.; Wu, Q.; Luo, X.; Li, D.; Lin, C.Y.; Yang, Y.; Qiu, L. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 1658–1677. <https://doi.org/10.18653/v1/2024.acl-long.91>.
 131. Pan, Z.; Wu, Q.; Jiang, H.; Xia, M.; Luo, X.; Zhang, J.; Lin, Q.; Rühle, V.; Yang, Y.; Lin, C.Y.; et al. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In *Findings of the Association for Computational Linguistics: ACL 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 963–981. <https://doi.org/10.18653/v1/2024.findings-acl.57>.
 132. Shandilya, S.; Xia, M.; Ghosh, S.; Jiang, H.; Zhang, J.; Wu, Q.; Rühle, V.; Rajmohan, S. TACO-RL: Task Aware Prompt Compression Optimization with Reinforcement Learning. In *Findings of the Association for Computational Linguistics: ACL 2025*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 1582–1597. <https://doi.org/10.18653/v1/2025.findings-acl.81>.
 133. Xu, F.; Shi, W.; Choi, E. RECOMP: Improving Retrieval-Augmented LMs with Context Compression and Selective Augmentation. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; pp. 43478–43502. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/hash/bda88ed2892f5e61c9a9bf215c566913-Abstract-Conference.html (accessed on 15 October 2026).
 134. Mu, J.; Li, X.L.; Goodman, N. Learning to Compress Prompts with Gist Tokens. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December 2023; Volume 36, pp. 19327–19352. <https://doi.org/10.52202/075280-0848>.
 135. Chevalier, A.; Wettig, A.; Ajith, A.; Chen, D. Adapting Language Models to Compress Contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 3829–3846. <https://doi.org/10.18653/v1/2023.emnlp-main.232>.
 136. Cheng, X.; Wang, X.; Zhang, X.; Ge, T.; Chen, S.Q.; Wei, F.; Zhang, H.; Zhao, D. xRAG: Extreme Context Compression for Retrieval-augmented Generation with One Token. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 109487–109516. <https://doi.org/10.52202/079017-3476>.
 137. Rau, D.; Wang, S.; Déjean, H.; Clinchant, S.; Kamps, J. Context Embeddings for Efficient Answer Generation in Retrieval-Augmented Generation. In Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining, Hannover, Germany, 10–14 March 2025; pp. 493–502. <https://doi.org/10.1145/3701551.3703527>.
 138. Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; et al. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December 2023; Volume 36, pp. 34661–34710. <https://doi.org/10.52202/075280-1506>.
 139. Liu, Z.; Desai, A.; Liao, F.; Wang, W.; Xie, V.; Xu, Z.; Kyrrilidis, A.; Shrivastava, A. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December 2023; Volume 36, pp. 52342–52364. <https://doi.org/10.52202/075280-2279>.
 140. Xiao, G.; Tian, Y.; Chen, B.; Han, S.; Lewis, M. Efficient Streaming Language Models with Attention Sinks. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 21875–21895. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/5e5fd18f863cbe6d8ae392a93fd271c9-Paper-Conference.pdf (accessed on 15 October 2026).
 141. Ge, S.; Zhang, Y.; Liu, L.; Zhang, M.; Han, J.; Gao, J. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 22975–22988. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf (accessed on 15 October 2026).
 142. Cai, Z.; Zhang, Y.; Gao, B.; Liu, Y.; Li, Y.; Liu, T.; Lu, K.; Xiong, W.; Dong, Y.; Hu, J.; et al. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. In Proceedings of the Conference on Language Modeling, Montreal, QC, Canada, 7–10 October 2025. Available online: <https://colmweb.org/2025/AcceptedPapers.html> (accessed on 15 October 2026).

143. Liu, A.; Liu, J.; Pan, Z.; He, Y.; Haffari, G.; Zhuang, B. MiniCache: KV Cache Compression in Depth Dimension for Large Language Models. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 10–15 December 2024; Volume 37, pp. 139997–140031. <https://doi.org/10.52202/079017-4443>.
144. Wan, Z.; Wu, X.; Zhang, Y.; Xin, Y.; Tao, C.; Zhu, Z.; Wang, X.; Luo, S.; Xiong, J.; Wang, L.; et al. D2O: Dynamic Discriminative Operations for Efficient Long-Context Inference of Large Language Models. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 87027–87052. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/hash/d862f7f5445255090de13b825b880d59-Abstract-Conference.html (accessed on 15 October 2026).
145. Saxena, U.; Saha, G.; Choudhary, S.; Roy, K. Eigen Attention: Attention in Low-Rank Space for KV Cache Compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 15332–15344. <https://doi.org/10.18653/v1/2024.findings-emnlp.899>.
146. Zhu, Y.; Yang, D.H.; Amiri, M.M.; Murugesan, K.; Pedapati, T.; Chen, P.Y. OjaKV: Context-Aware Online Low-Rank KV Cache Compression. In *Findings of the Association for Computational Linguistics: ACL 2026*; Association for Computational Linguistics: Stroudsburg, PA, USA, San Diego, CA, USA, 2–7 July 2026; pp. 10161–10178. <https://doi.org/10.18653/v1/2026.findings-acl.494>.
147. Sun, H.; Chang, L.W.; Bao, W.; Zheng, S.; Zheng, N.; Liu, X.; Dong, H.; Chi, Y.; Chen, B. ShadowKV: KV Cache in Shadows for High-Throughput Long-Context LLM Inference. In Proceedings of the 42nd International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 57355–57373. Available online: <https://proceedings.mlr.press/v267/sun25b.html> (accessed on 15 October 2026).
148. Zhang, Z.; Liu, S.; Chen, R.; Kailkhura, B.; Chen, B.; Wang, Z. Q-Hitter: A Better Token Oracle for Efficient LLM Inference via Sparse-Quantized KV Cache. In Proceedings of Machine Learning and Systems, Santa Clara, CA, USA, 13–16 May 2024; Volume 6, pp. 381–394. Available online: https://proceedings.mlsys.org/paper_files/paper/2024/file/bbb7506579431a85861a05fff048d3e1-Paper-Conference.pdf (accessed on 15 October 2026).
149. Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; et al. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 3119–3137. <https://doi.org/10.18653/v1/2024.acl-long.172>.
150. Belikova, J.; Rozhevskii, D.; Svirin, D.; Polev, K.; Panchenko, A. Detecting Overflow in Compressed Token Representations for Retrieval-Augmented Generation. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2026; pp. 797–810. <https://doi.org/10.18653/v1/2026.eacl-srw.59>.
151. Zhang, C.; Cheng, J.; Constantinides, G.A.; Zhao, Y. LQER: Low-Rank Quantization Error Reconstruction for LLMs. In Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024; Volume 235, pp. 58763–58779. Available online: <https://proceedings.mlr.press/v235/zhang24j.html> (accessed on 15 October 2026).
152. Dettmers, T.; Svirschevski, R.; Egiazarian, V.; Kuznedelev, D.; Frantar, E.; Ashkboos, S.; Borzunov, A.; Hoefler, T.; Alistarh, D. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 5733–5761. Available online: https://proceedings.iclr.cc/paper_files/paper/2024/file/1787533e171dcc8549cc2eb5a4840eec-Paper-Conference.pdf (accessed on 15 October 2026).
153. Li, Y.; Yu, Y.; Zhang, Q.; Liang, C.; He, P.; Chen, W.; Zhao, T. LoSparse: Structured Compression of Large Language Models based on Low-Rank and Sparse Approximation. In Proceedings of the 40th International Conference on Machine Learning, Honolulu, HI, USA, 23–29 July 2023; Volume 202, pp. 20336–20350. Available online: <https://proceedings.mlr.press/v202/li23ap.html> (accessed on 15 October 2026).
154. Yao, J.; Gultepe, E. SPQ: An Ensemble Technique for Large Language Model Compression. In *Proceedings of the Fifteenth Language Resources and Evaluation Conference, Palma, Mallorca, Spain, 11–16 May 2026*; ELRA Language Resources Association: Paris, France, 2026; pp. 6185–6195. <https://doi.org/10.63317/3t98siww4xf2>.
155. Kang, M.; Jeong, J.; Lee, S.; Cho, J.; Hwang, S.J. Distilling LLM Agent into Small Models with Retrieval and Code Tools. In Proceedings of the Advances in Neural Information Processing Systems, San Diego, CA, USA, 2–7 December 2025; Volume 38, pp. 106501–106538. <https://doi.org/10.52202/085713-3553>.

156. Kang, M.; Chen, W.N.; Han, D.; Inan, H.A.; Wutschitz, L.; Chen, Y.; Sim, R.; Rajmohan, S. ACON: Optimizing Context Compression for Long-horizon LLM Agents. *arXiv* **2025**, arXiv:2510.00615. <https://doi.org/10.48550/arXiv.2510.00615>.
157. Zhang, G.; Yue, Y.; Li, Z.; Yun, S.; Wan, G.; Wang, K.; Cheng, D.; Yu, J.X.; Chen, T. Cut the Crap: An Economical Communication Pipeline for LLM-based Multi-Agent Systems. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 75389–75428. Available online: https://proceedings.iclr.cc/paper_files/paper/2025/hash/bbc461518c59a2a8d64e70e2c38c4a0e-Abstract-Conference.html (accessed on 15 October 2026).

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.