

TOWARDS NEXT-GENERATION ARCHITECTURES OF LARGE LANGUAGE MODELS: A SURVEY

Yao Teng^{1*}, Guoxuan Chen^{1*}, Yimeng Chen^{2†}, Han Shi^{3†},
Piotr Piękos², Jie Ran³, Yutao Zeng⁴, Chuanyang Zheng³, Jingyao Li⁵, Li Nanbo²,
Haoli Bai³, Chao Huang¹, Xihui Liu¹, Di He⁶, Weiyang Liu^{5,7}, Jürgen Schmidhuber^{2,8}

¹HKU ²Center of Excellence for Generative AI, KAUST ³Huawei Technologies ⁴ByteDance Seed
⁵CUHK ⁶Peking University ⁷Max Planck Institute for Intelligent Systems ⁸The Swiss AI Lab, IDSIA-USI/SUPSI

<https://surveyllmarch.github.io>

ABSTRACT

Large Language Models (LLMs) have advanced rapidly through increases in model scale, data, and computation, while their underlying architectures have also undergone substantial evolution. Beyond the canonical Transformer architecture, recent designs seek greater efficiency, scalability, and flexibility by rethinking how sequence information is processed, represented, and transformed. We survey efficient attention and state management, modern recurrent and other linear-complexity sequence models, Mixture-of-Experts architectures, and advances in activation functions, normalization, residual connections, positional encoding, and tokenization. We also cover multi-token prediction and adaptive and recurrent depth. Across these areas, we highlight key design principles and trade-offs among representational power, optimization stability, and computational cost. Complementing this component-wise treatment, we take a system-level perspective. We trace representative open-weight LLM families to identify recurring patterns in how architectural choices are combined, and synthesize these developments along four dimensions—processing structure, transformation dynamics, state management, and output commitment—characterizing how recent designs preserve, refine, or relax the assumptions of the canonical decoder. Together, this survey aims to provide a systematic overview of LLM architectural evolution and to chart pathways toward next-generation designs.

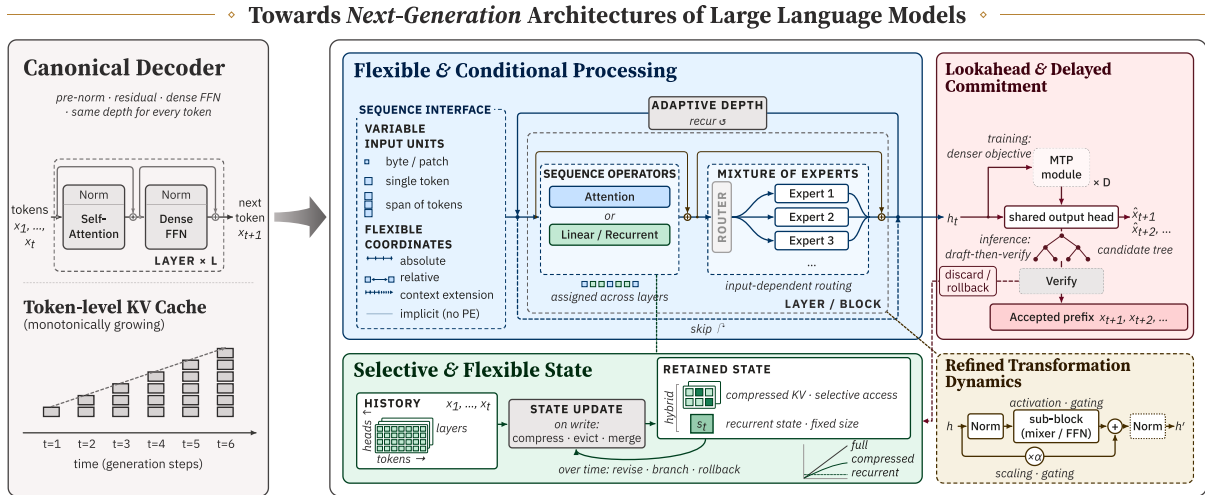


Figure 1: Overview of the recent evolution of large language model architectures. Section 15 develops this view.

*Equal Contribution. †Corresponding Authors.

Contents

1	Introduction	5
2	Overview	8
3	History of Language Modeling Architectures	10
4	Basic Modules in Transformer	12
4.1	Multi-head Attention (MHA)	12
4.2	Feed-Forward Network	13
4.3	Positional Encoding	14
4.4	Residual Connections and Normalization	15
4.5	Tokenization and Input Embeddings	15
4.6	Output Projection and Autoregressive Generation	15
4.7	Encoder/Decoder Transformer Configurations	16
4.7.1	Decoder-only	16
4.7.2	Encoder-only	16
4.7.3	Encoder-Decoder	16
4.8	Computational Complexity	16
5	Efficient Attention and KV Cache Management	18
5.1	Token Reduction	18
5.2	Head Reduction	25
5.3	Dimension Reduction	27
5.4	Layer Reduction	28
5.5	Future Directions	29
6	Linear Language Models	29
6.1	Recurrent Neural Networks	29
6.2	Fast Weight Programmers	31
6.3	State Space Models	32
6.4	Linearized Attention	33
6.5	Unification and Comparison of Linear Operators	35
6.6	Future Directions	39
7	Mixture of Experts	40
7.1	Evolution of MoE in LLMs	40
7.2	Future Directions	44
8	Activation Functions	45
8.1	Evolution of Activation Functions in LLMs	45
8.2	Future Directions	46
9	Normalization and Residual Connections	47
9.1	Normalization Methods	48
9.2	Combinations of Normalizations and Residual Connections	48
9.3	Future Directions	51
10	Positional Encoding	51
10.1	Absolute Positional Encoding	51
10.2	Relative Positional Encoding	54
10.3	Context Extension via Positional Adaptation	55
10.4	Future Directions	57
11	Tokenization	57
11.1	Tokenizers with Different Granularities	57
11.2	Vocabulary Size	58
11.3	Future Directions	60
12	Multi-Token Prediction	60

12.1	Parallel Prediction	61
12.2	Sequential Prediction	62
12.3	Inference Integration	63
12.4	Future Directions	63
13	Adaptive and Recurrent Depth	64
13.1	Depth Reduction and Early Exit	64
13.2	Conditional Depth over Distinct Layers	65
13.3	Looped (Depth-Recurrent) Transformers	65
13.4	Training and Scaling Variable Depth	66
13.5	Future Directions	67
14	Popular Architectures	67
14.1	Evolution of Open-Weight LLMs and Their Precursors	67
14.2	Recurring Architectural Design Patterns	68
14.2.1	Canonical Dense Decoders	68
14.2.2	Sparse Capacity with Compressed Attention State	69
14.2.3	Hybrid Sequence and State Processing	69
14.2.4	Prediction-Augmented Backbones	70
15	Beyond Components: A Unified View of Architectural Design	79
15.1	The Canonical Computational Structure and Its Implicit Design Assumptions	79
15.2	Design Directions Beyond the Canonical Decoder	79
15.2.1	Flexible and Conditional Processing	79
15.2.2	Refined Transformation Dynamics	81
15.2.3	Selective and Flexible State Management	81
15.2.4	Lookahead and Delayed Commitment	81
15.3	Interdependencies and Joint Design	82
16	Concluding Remarks	83

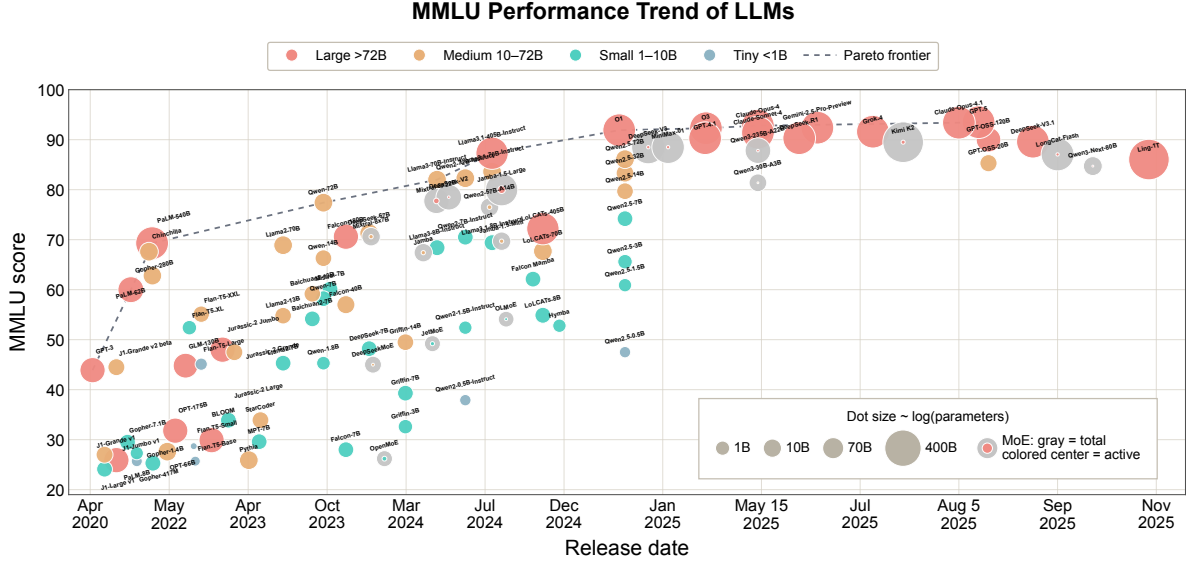


Figure 2: **MMLU performance versus release date and parameter count.** X-axis: release date; y-axis: MMLU performance [157]. Dot sizes of older closed-source models are illustrative (parameter counts undisclosed). The trends visible in this figure are discussed in Sec. 1.

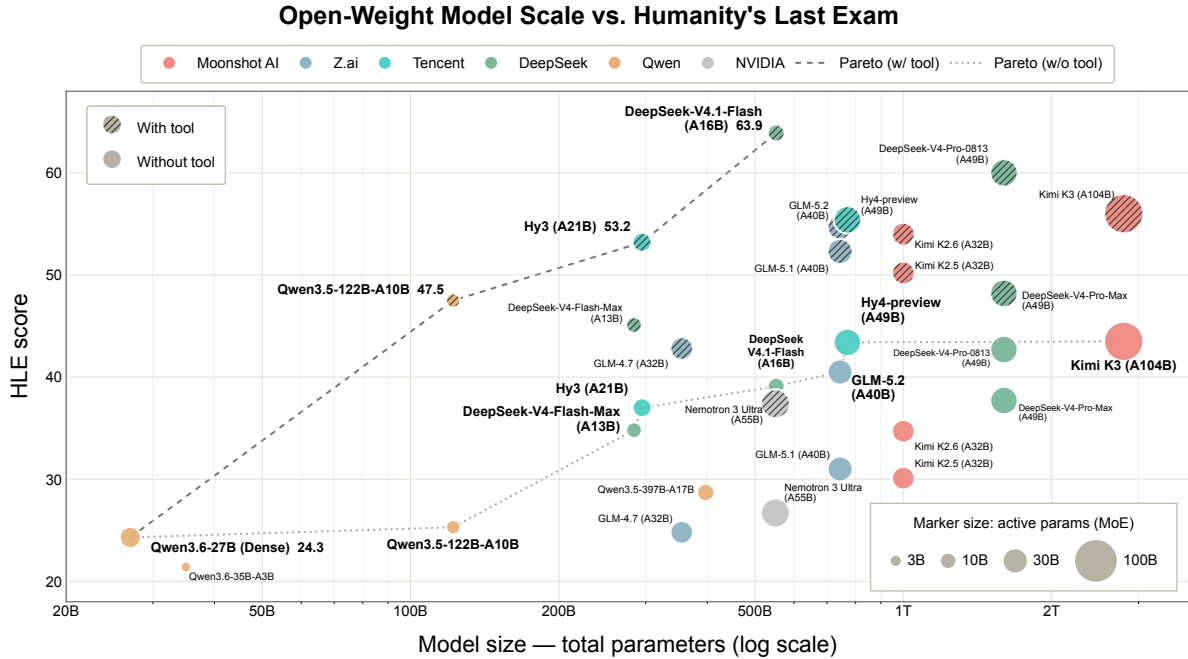


Figure 3: **Humanity’s Last Exam (HLE) [40] performance versus open-weight model size.** The figure primarily covers newly released 2026 models; GLM-4.7, released in late 2025, is retained as a reference. The reported values are based primarily on the Open LLM Leaderboard [280] and the HLE leaderboard [41]. The horizontal axis shows total parameters on a logarithmic scale, while marker area increases nonlinearly with active parameters for MoE models and total parameters for dense models. Colors denote developers. The dashed and dotted lines trace the corresponding observed Pareto frontiers; both start from Qwen3.6-27B (Dense), whose score is taken from [374], although its tool-use protocol is unspecified. Because the figure mixes tool-enabled and non-tool protocols, scores should be compared with caution.

1 Introduction

Large language models (LLMs) have revolutionized artificial intelligence research and applications, demonstrating remarkable capabilities across a wide range of tasks. As one of the most important foundation models, they now serve as the backbone of modern AI systems, driving progress in language understanding, generation, reasoning, and multimodal applications. The success of these models is grounded in decades of research in language modeling (Sec. 3), culminating in architectures capable of learning from massive text corpora and exhibiting emergent abilities, such as advanced reasoning, code generation, and complex problem solving. This progress has also benefited from substantial long-run reductions in the cost of computation [329, 65]², making it economically feasible to train models that would have been prohibitively expensive just a decade ago.

A distinctive feature of modern LLM development is that capabilities tend to improve systematically when models are trained with larger parameter counts and data. This has pushed current practice toward a regime in which scaling is a central means of obtaining stronger language models. In this large-scale regime, contemporary LLMs commonly contain tens to hundreds of billions of parameters, are trained on hundreds of billions to trillions of tokens, and require substantial compute budgets. Across such scales, empirical scaling laws reveal relatively smooth relationships between model size, training data, compute, and achievable loss [217, 168]. Across successive generations, a combination of increased scale, improved training recipes, and refined architectures has led to steady gains on standardized benchmarks such as MMLU (Massive Multitask Language Understanding) [157] (Figure 2), GPQA (Graduate-Level Google-Proof Q&A) [385], and SWE-bench (Software Engineering Benchmark) [208], reflecting advances in broad language understanding, expert-level reasoning, and practical coding ability.

The remarkable success of large language models across diverse applications has created an urgent demand for improved computational efficiency and scalability. As models continue to grow in size and are deployed to tasks requiring longer context windows, the computational costs for both training and inference have emerged as a major bottleneck. Balancing efficiency and stability in large-scale training, while simultaneously advancing model performance, remains a fundamental challenge for the field. At the core of these challenges is the model *architecture*, which dictates computational complexity, optimization dynamics, and representational capacity.

While early pretrained language models were based on Long Short-Term Memory (LSTM) networks [167, 166] (e.g., ELMo [357], ULMFiT [171]), architecturally, most contemporary LLMs instantiate variations of the modern Transformer architecture proposed by Vaswani et al. in 2017 [489], whose core attention mechanism traces back to the Unnormalized Linear Transformer (ULTRA) [406, 408, 420, 419] proposed in 1991 by the same lab that introduced LSTM. A standard Transformer stack, as will be introduced in Sec. 4, typically consists of multi-head self-attention, position-wise feed-forward network, positional encoding, residual connection, and normalization. Tokenization further defines the interface between raw text and model inputs. Despite its empirical success and dominant status, the modern Transformer architecture exhibits inherent inefficiencies that become increasingly pronounced at scale, underscoring the need for targeted improvements to its core components. The primary source of inefficiency, especially for long sequences, is the quadratic computational complexity of the self-attention mechanism [465], $O(T^2)$ with respect to sequence length T , together with the linear growth of the key–value (KV) cache [432, 147] during inference, which heavily strains memory resources. These limitations have motivated extensive research into alternative architectures.

Recent efforts to address them can be broadly categorized into two directions. The first category comprises **Efficient Attention and KV Cache Management**. These methods preserve the core structure of quadratic attention while reducing the effective computational cost and memory usage (especially regarding the growth of the KV cache) by decreasing the number of key–value pairs involved in the computation. Efficient attention methods can be organized along four axes of KV storage and access, with different implications for attention computation: token-oriented [46, 547, 282, 519], head-oriented [432, 1], dimension-oriented [73], and layer-oriented [270, 591]. Token-oriented methods use fixed or input-dependent selection, eviction, or merging to reduce attended or stored entries; head-oriented methods share KV heads among query heads; dimension-oriented methods compress cached representations; and layer-oriented methods share KV states across layers. These changes primarily target storage and memory traffic, and only some also reduce the quadratic arithmetic cost of dense attention.

²Nordhaus [329] reports an average logarithmic improvement rate of approximately 45% per year in constant-dollar computational performance since 1945, using data extending to 2006. Coyle & Hampton [65] estimate a 100–300× reduction in computation costs since 2000, depending on the index construction.

The second category, which we refer to as **Linear Language Models**, seeks linear or near-linear complexity through a more fundamental architectural shift. Instead of modifying or sparsifying quadratic attention, these approaches adopt alternative sequence modeling mechanisms that avoid explicit pairwise token interactions, thereby achieving intrinsically linear scaling. The technical roots of linear language models can be traced back to fast weight programmers (FWPs) [406, 408, 416] and LSTM. ULTRA in 1991 represents the early linear architecture, which is an FWP with the additive outer-product fast-weight principle. This principle resurfaces in modern linear attention: Katharopoulos et al. [219] proposed kernel-based mechanisms to approximate quadratic attention, which are mathematically equivalent to ULTRA equipped with a feature map on queries and keys and a running normalizer [403], and further highlighted the close connection between linear attention and linear recurrent neural network (RNN) formulations. LSTM is an influential precursor on the recurrent side: it maintains a bounded, constant-size state updated additively under an input-dependent decay [117], from which outputs are read (its *constant-error carousel* [167] can also be seen as the residual connection which is now ubiquitous in deep networks [444, 152, 415, 418]). Recent research has introduced linear recurrent models derived from RNNs [343, 352] and LSTM [24], FWPs [403, 540, 541, 354], and state space models [135, 136, 70]. Despite their diverse appearance and origin, these models share a common recurrent skeleton such that their core linear operators can be analyzed within a unified framework [540, 498]. Notably, the recurrent state update in some of these models can be interpreted as the online update mechanism in FWPs: each new token provides a key-value association, and the model immediately updates its state during the forward pass, analogous to analytic online regression or one step of online gradient descent [541, 498]. This form of test-time adaptation was already present in the 1991 FWP/ULTRA [406, 408]. Current research actively investigates both the representational capacity in linear models [294, 131] and the development of hybrid architectures [72, 24, 12, 321, 541, 196] that combine linear operators with quadratic attention to balance efficiency and expressiveness.

Beyond the attention mechanism, feed-forward network (FFN) layers represent another critical target for improvement. As the primary channel-mixing component and a major repository of model knowledge, FFNs can dominate the computational load when sequence length is not the primary factor. To address this, the **Mixture of Experts (MoE)** [144, 200, 211] architecture has emerged as a promising technique for scaling model capacity without a proportional increase in computational cost [434, 246, 96]. Sparse MoE replaces selected FFNs with multiple expert sub-networks and routes each token to a small subset (e.g., top-2). Only the selected experts are evaluated for that token, so expert computation scales with active rather than total expert capacity. This enables a larger total parameter capacity at a comparable per-token expert-compute budget [62]. Realized training and inference speed also depends on routing, communication, load balance, and implementation. Recent efforts focus on routing strategies [96, 248, 579], module integration [382, 246, 68, 563], and training stability [590], with successful adoption in several large-scale models [515, 313, 68, 435, 320].

Another critical component within the FFN layers is the **Activation Function**, which provides the non-linearity enabling models to learn complex patterns [142]. Activation designs include the ReLU/ELU family, such as Rectified Linear Unit (ReLU) [105, 106] and Scaled Exponential Linear Unit (SELU) [231], and smooth functions such as Gaussian Error Linear Unit (GELU) and Sigmoid Linear Unit (SiLU) [156, 92]. Swish [383] equals SiLU when its sigmoid-scale parameter is one. A parallel development, Gated Linear Units (GLU) [71], adapted multiplicative gating from LSTM [167] to feed-forward networks. SwiGLU [433] combines an explicit dual-path FFN with a SiLU gate and has been widely adopted following strong empirical results for the complete gated design.

The continuous evolution of the Transformer architecture also encompasses several critical auxiliary components. **Positional Encoding (PE)** supplies explicit order information. Full self-attention without positional or other order-dependent signals is permutation-equivariant; causal masking also introduces ordering constraints, so explicit PE is not the only possible source of sequence order. Absolute Positional Encoding (APE) is adopted in the early Transformers based on dot-product attention, such as sinusoidal functions [489] or learnable embeddings [378, 80]. However, APE often struggles with length extrapolation. Relative Positional Encoding (RPE) represents displacement between tokens directly within attention, as in the relative key and value representations of Shaw et al. [431]. Later methods realize relative-position dependence through different mechanisms, including rotations of queries and keys [450] and other position-dependent transformations [127]. Rotary Position Embedding (RoPE) [450], a new form of relative positional encoding, has emerged as an advanced choice in LLMs like LLaMA [486, 295], owing to its mathematical elegance and strong empirical effectiveness. Ongoing research continues to explore more flexible and adaptive PE techniques capable of extrapolating to longer context windows [48, 82, 355, 51]. In addition, **Residual Connections**, exemplified by Highway Networks and ResNets [444, 152], facilitate the optimization of deep networks [443, 523, 496]. LSTM

provides an earlier example of a protected additive state pathway with multiplicative gates [167], while the forget gate was introduced subsequently [117]. **Normalization** mechanisms [183, 13] help control activation scales and support optimization. Many recent models, including LLaMA, use RMSNorm [555, 486], which omits the mean-centering step of LayerNorm. Current research develops more efficient normalization methods [207, 328] and studies their interplay with residual connections [523, 81, 252, 336], particularly in balancing representational capacity with training stability.

Finally, the interface between a neural network and raw text data, **Tokenization**, represents another crucial component. Although often treated as a preprocessing step, tokenization directly impacts both the efficiency and capability of an LLM [358]. The choice of granularity, ranging from characters to words to the now-dominant subword schemes, introduces a fundamental trade-off among vocabulary size, sequence length, and linguistic expressiveness [299]. Recent studies have identified vocabulary size itself as a key scaling dimension, demonstrating that its optimization can yield performance gains comparable to scaling model parameters, but at reduced computational cost [464, 176]. Building on this, techniques such as decoupling input and output vocabularies [176, 544] and progressively expanding the vocabulary during pre-training [545] turn the vocabulary and input interface into design variables rather than fixed preprocessing choices. In parallel, a complementary line of research explores tokenization-free models that operate directly on raw bytes, aiming to bypass fixed-vocabulary constraints entirely to foster more universal systems [531, 344] capable of handling open-ended data streams.

Beyond the design of individual layers, two more recent lines of work revisit how a decoder allocates computation along the two axes of generation: how many tokens it predicts per forward pass, and how many layers each token traverses. **Multi-Token Prediction (MTP)** extends the standard next-token objective so that each position is trained to predict several future tokens [446, 362, 123]. This densifies the training signal and encourages representations that encode longer-horizon information; moreover, the additional predictors can be reused at inference as cheap drafts of the next few tokens, which the model verifies in a single parallel forward pass under the draft-then-verify paradigm of speculative decoding [247, 35, 256]. Architectural designs range from parallel prediction heads on a shared backbone [123, 35] to sequential predictors that preserve the causal chain among future tokens [269, 256], and sequential MTP has rapidly become a standard component of recent open-weight LLMs [269, 278, 479, 477]. In a complementary direction, **Adaptive and Recurrent Depth** architectures relax the assumption that every token must pass through a fixed stack of layers. Early-exit and layer-reduction methods terminate computation once an intermediate representation is sufficiently predictive [91, 425, 95, 93]; conditional-depth routing such as Mixture-of-Depths lets only a subset of tokens execute a given block [384]; and looped, or depth-recurrent, Transformers repeatedly apply a shared block to an evolving latent state, adding computational depth without adding parameters [77, 111, 587, 16]. These designs turn depth from a fixed hyperparameter into an allocatable resource, while raising their own questions about supervising intermediate states, maintaining dynamical stability under repeated computation, and quantifying the exchange rate between parameter capacity and repeated computation [426].

Several existing surveys cover parts of this landscape, but typically from the perspective of a single component or a single objective. Component-specific surveys address efficient and sparse attention [465], KV cache management [147], mixture of experts [36, 319], state space models [369, 350], positional encoding and length extrapolation [573], activation functions [142, 87], and speculative decoding [517]; broader surveys of LLMs [574] and of efficient LLMs [492, 582] treat architecture as one topic among training, data, alignment, and serving. This survey differs in three respects. First, it covers the full set of architectural components of a modern decoder, from tokenization and positional encoding to attention, FFNs, normalization, residual pathways, output objectives, and depth allocation, so that design decisions can be compared under a common notation. Second, it pairs the component-wise review with a system-level view that traces how open-weight model families actually combine these components (Sec. 14). Third, it offers a unified reading of these developments in terms of the implicit assumptions of the canonical decoder that each of them preserves or relaxes (Sec. 15).

The central aim of this survey is to systematically examine the diverse architectural innovations that aspire to shape the *next generation* of Large Language Models. The following sections explore these emerging paradigms, spanning efficient attention mechanisms, linear language models, conditional computation strategies such as mixture of experts, advances in core components, multi-token prediction, and adaptive depth. We carefully examine their potential to overcome current limitations and reshape the future landscape of AI. The system-level and unified perspectives of Secs. 14 and 15 then organize these developments along four broad directions: flexible and conditional processing, refined transformation dynamics, selective and flexible state management, and lookahead and delayed commitment.

This perspective emphasizes that modern architectures are better understood as joint configurations of interdependent design dimensions than as collections of independently chosen modules.

To situate these developments empirically, we close this introduction with two snapshots that together span the recent history of LLMs. Figure 2 covers the earlier generations, released between 2021 and 2025, by plotting MMLU performance [157] against release date and parameter count, and four trends stand out. Consistent with empirical scaling laws [217, 168], larger models (larger bubbles) reliably outperform their smaller contemporaries; at the same time, improvements in data quality, optimization, and architecture let newer, smaller models frequently surpass older and much larger ones, reflecting a shift toward compute-optimal, data-rich training regimes. MoE architectures emerge as a prevailing design, reaching top-tier performance even at high sparsity while offering substantial inference-efficiency benefits [96], and a clear stratification separates proprietary from open-weight models, with proprietary systems (e.g., the GPT and Claude series) consistently defining the performance upper bound. By 2026, however, MMLU was approaching saturation, so Figure 3 turns to the most recent generation of open-weight models, released mainly in 2026, and evaluates them on Humanity’s Last Exam (HLE) [40], a far more discriminative benchmark of frontier capabilities. The picture extends the earlier trends: nearly all of the plotted models are sparse MoE architectures, HLE scores still rise with total parameter count along the observed Pareto frontiers, and tool use adds well over ten points for the strongest systems. The gap to proprietary models also persists: the best plotted open-weight model reaches 56.0% with tools, whereas Anthropic’s latest limited-access model, Claude Mythos 5 [10], scores 59.0% without tools and 64.5% with tools. Because reported HLE results mix tool-enabled and non-tool protocols, however, these scores should be compared with caution.

2 Overview

This survey focuses on the architectural design of autoregressive language-model backbones, including sequence processing, parameter activation, representation and state management, and prediction and depth allocation. Earlier language models and related architectures are included where they clarify the origins or mechanisms of contemporary designs. Training objectives, decoding procedures, and systems techniques are discussed insofar as they shape architectural choices or determine their practical costs. The literature covered extends through August 2026. We select representative methods based on their introduction of distinct mechanisms, their role in clarifying important design trade-offs, or their adoption in documented model families. This selection includes foundational approaches, influential variants, and alternative designs that expose limitations of prevailing choices.

Our discussion draws on original research papers, technical reports, and official implementation materials. Research papers provide the primary basis for mathematical formulations, theoretical claims, and experimental findings, with conclusions interpreted under their stated assumptions and evaluation settings. Technical reports document model-scale configurations, training recipes, and reported results; these observations establish practical use but generally do not isolate the contribution of individual components. Official code and released configurations clarify implementation details and variant-specific settings that may be omitted from the text. Where sources or versions differ, we identify the configuration being discussed. Throughout the survey, we distinguish formal results, empirical findings, documented adoption, and our own synthesis of architectural trends.

Figure 4 provides a detailed roadmap of this survey. Section 3 first traces the history of language modeling architectures, from statistical n -gram models through recurrent networks and fast weight programmers to the Transformer. We then revisit in Sec. 4 the modern Transformer architecture and its core building blocks, including self-attention, feed-forward networks, and positional encoding. Next, we turn to methods that optimize the self-attention module: Sec. 5 surveys efficient attention and key–value (KV) cache management approaches that reduce computation and memory along four axes: token reduction, head reduction, dimension reduction, and layer reduction; Sec. 6 focuses on linear language models that approximate or substitute the attention mechanism to reduce its complexity from quadratic to linear. We introduce mixture of experts in Sec. 7, tracing three overlapping developments from sparsely gated FFNs through fine-grained shared-expert designs to systems-oriented variants; we then review the evolution of activation functions in Sec. 8, and discuss residual connections and normalization techniques in Sec. 9. Positional encoding variants are examined in Sec. 10, while related work on tokenization is covered in Sec. 11. The next two sections turn from individual layers to how computation is allocated during generation. Section 12 discusses multi-token prediction, classifying methods by how the K future tokens are factored: parallel prediction heads that are conditionally independent given the backbone representation, and sequential predictors that preserve the causal chain among future tokens. A third group,

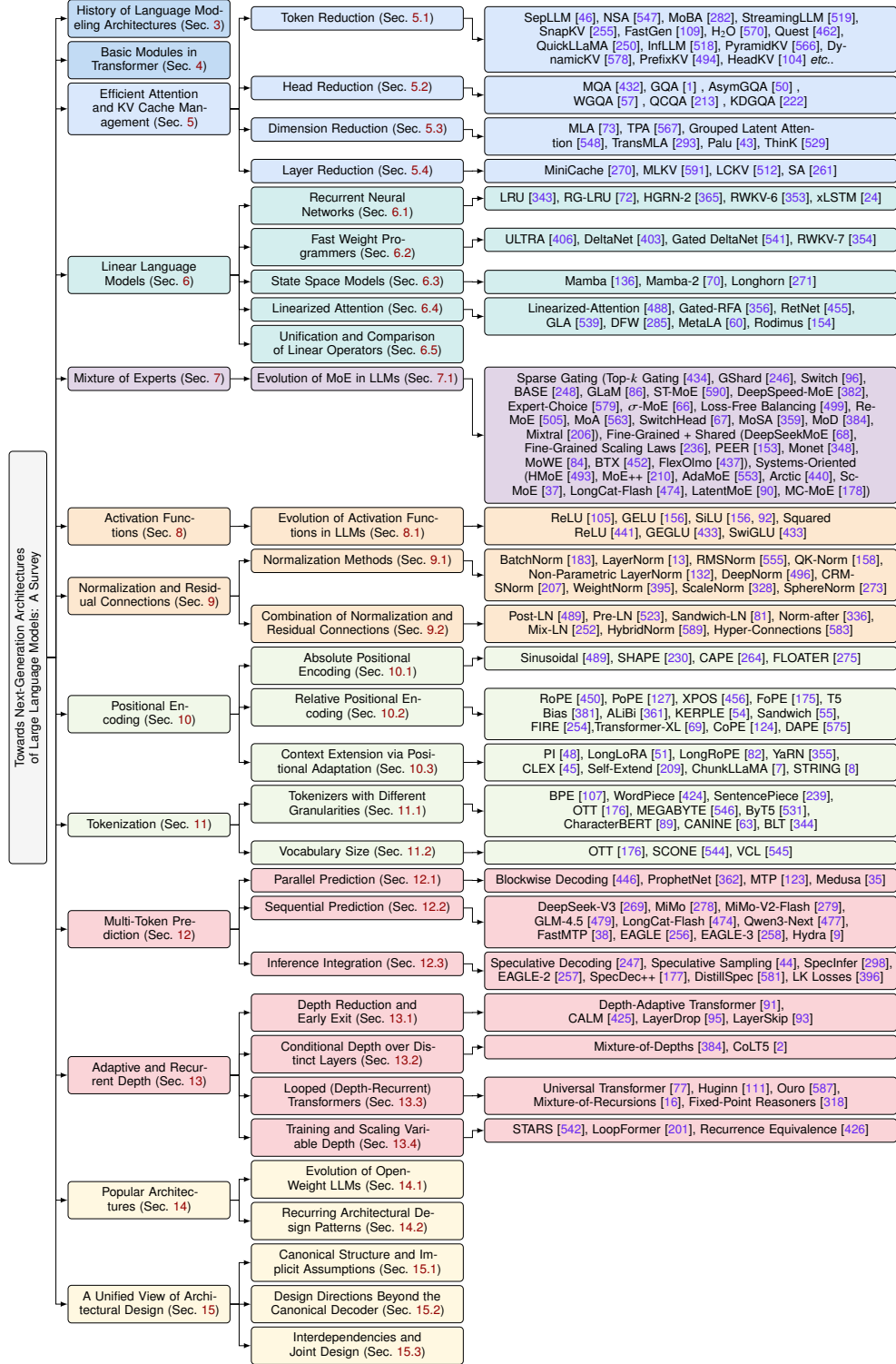


Figure 4: Taxonomy of architectures for large language models. Colors group sections by role in the stack: mid-blue is the historical background and Transformer foundation; light blue is efficient attention and KV cache management and teal is linear sequence operators (two ways to cheapen mixing over time); purple is mixture of experts; peach is intra-block transforms (activation and normalization); green is the input interface (tokenization and position); coral is how much compute is spent per generated token (multi-token prediction and adaptive depth); cream is system-level synthesis.

inference integration, cuts across both factorizations: it collects methods that improve how drafted tokens are verified and accepted under draft-then-verify, through acceptance rules, tree-structured verification, adaptive draft lengths, and drafter alignment. Section 13 reviews adaptive and recurrent depth, covering early exit and layer reduction, conditional depth over distinct layers, looped (depth-recurrent) Transformers, and the training and scaling issues that arise when executed depth becomes variable. In Sec. 14, we synthesize these perspectives at the system level: we first trace the evolution of open-weight LLM families and their precursors, including the LLaMA series [486, 487, 295, 296], Qwen series [20, 476, 370, 371], and DeepSeek series [30, 73, 68, 269], and then identify recurring architectural design patterns that cut across families, namely canonical dense decoders, sparse capacity with compressed attention state, hybrid sequence and state processing, and prediction-augmented backbones. Section 15 then offers a unified view of architectural design that goes beyond individual components: it makes explicit the implicit assumptions of the canonical decoder, organizes the surveyed mechanisms into four broad directions (flexible and conditional processing, refined transformation dynamics, selective and flexible state management, and lookahead and delayed commitment), and examines the interdependencies that make joint design necessary. We conclude this survey in Sec. 16.

3 History of Language Modeling Architectures

The transformative power of modern large language models did not emerge in isolation; it is the result of a decades-long pursuit to address the fundamental challenge of language modeling: developing statistical models that can assign probabilities to arbitrary sequences of words. Understanding this historical trajectory illuminates both the foundations upon which current LLMs are built and the recurring themes that continue to drive architectural innovation.

Contemporary generative language models are fundamentally grounded in viewing natural language as a stochastic process. In this formulation, random variables take values from a finite alphabet (characters, words, or subword tokens) and natural sentences are interpreted as samples drawn from these random sequences. The central task of a language model is to estimate transition probabilities, namely the conditional probability of the next symbol given a preceding context, thereby emulating the generative dynamics of language. This probabilistic paradigm traces back to foundational work in the early twentieth century. Prior to modern information theory, A. A. Markov established the mathematical basis for modeling dependent sequences; his empirical analysis of vowel-consonant patterns in literary text provided the first quantitative evidence that language exhibits non-trivial conditional probability structure [286]. Building on this insight, Claude Shannon explicitly proposed modeling language as a stochastic process that generates finite symbol sequences, formally equating it with a discrete information source [430]. By using n -gram models, such as digrams and trigrams, to capture what he termed “adjacent dependencies”, Shannon demonstrated how the choice of a symbol depends on its preceding history, and validated this perspective by generating increasingly realistic approximations of English text. His subsequent investigations into the redundancy and compressibility of English further established the theoretical feasibility of statistical language modeling [429].

Following Shannon’s paradigm, early statistical language models (SLM) primarily relied on finite-order Markov assumptions, estimating conditional distributions for discrete n -gram contexts. This stochastic perspective was formally operationalized for continuous speech recognition by Jelinek [203], and rigorously formulated as a maximum likelihood problem by Bahl et al. [18]. By modeling the speaker as a text generator (source) and the acoustic processor as a noisy channel, these works established the “Language Model” as a critical prior probability component in the linguistic decoder [203, 18]. To address the inherent data sparsity that assigns zero probability to unseen contexts, early systems incorporated various smoothing techniques, such as discounting [125, 509], back-off [220, 326, 234], and interpolation [204]. By the 1990s, research expanded to address the problem of “data fragmentation” inherent in high-order n -grams, introducing structured generalization mechanisms like class-based models [33, 233], decision trees [19], and Maximum Entropy frameworks [79, 242, 28]. Finite-order n -grams remained limited to local contexts, whereas trigger-based Maximum Entropy models and related approaches attempted to incorporate longer-range information. Efficiently capturing complex dependencies and deeper linguistic structure nevertheless remained difficult. As noted by Rosenfeld [392], the field faced a persistent tension: while “knowledge-impovertished” n -gram techniques proved remarkably robust and data-optimal, attempts to incorporate deeper linguistic structure often struggled to overcome the trade-off between computational tractability and the effective modeling of complex dependencies.

A fundamental shift emerged with the introduction of distributed representations [393, 162, 427, 323, 360] into language modeling [422, 27]. By embedding linguistic units into shared continuous latent spaces, these models captured semantic and syntactic regularities implicitly through parameter sharing, rather than through manually engineered back-off rules

or smoothing heuristics. In this way, generalization was internalized within the model itself, encoded as inductive biases induced by continuous, parameterized neural architectures. Early work in 1996 [422] demonstrated the viability of this paradigm in character-level language modeling with feed-forward neural networks (FFNs), showing the advantage of neural approaches in text compression compared with statistical methods; the lineage of such networks runs from perceptrons and early multilayer learning systems [390, 212, 391, 445, 198, 199, 161] (see Schmidhuber [414] for a review). Subsequent research [27] extended to word-level modeling, further establishing the potential of neural language models. From this point onward, neural architectures gradually displaced traditional statistical models and became the dominant paradigm in language modeling. Progress thereafter became increasingly tied to architectural innovations aimed at balancing representational capacity, optimization stability, and computational efficiency.

Subsequent progress in language modeling was driven largely by architectural innovations. Language-modeling research expanded from feed-forward networks to recurrent neural networks (RNNs) [94, 301], a transition rooted in a long historical context. Unlike feed-forward architectures, RNNs incorporate a memory mechanism via feedback connections that allows information from previous time steps to influence current predictions [405]. The conceptual foundations of recurrent computation can be traced back to non-learning dynamical systems from the 1920s, such as the *Ising model* (or Lenz-Ising model) [245, 197, 237, 506], which can be viewed retrospectively as early models of interacting units with cyclic dependencies, rather than as recurrent sequence models with learned temporal memory. Later, foundational works [289, 232] proved that such recurrent structures could function as formal computational units capable of processing sequences. Work on trainable feedback networks around the early 1970s included adaptive association and learning rules [6, 324]; Hopfield’s associative-memory networks followed in 1982 [170]. Related studies examined biological neural dynamics [133] and hippocampal memory [287]. Recurrent models were used to investigate linguistic sequence structure around 1990 [94], and were also applied to acoustic modeling for speech recognition [388] and recursive translation [100]. Recurrent neural language models subsequently gained prominence through work such as Mikolov et al. [301, 302, 300] and Graves [128]. However, while RNNs can in principle process sequences of arbitrary length, the vanishing and exploding gradient problems [164, 165] make it difficult for them to capture long-range dependencies in practice.

Long Short-Term Memory (LSTM) networks [167, 166] address a central aspect of the vanishing-gradient problem analyzed in earlier work [165]. Their constant error carousels (CECs) preserve error flow along a designated memory-state pathway; this does not guarantee that every gradient in the full network avoids vanishing or explosion. The protected additive pathway offers a structural comparison with later gated skip and residual connections [444, 152], without making the mechanisms historically or mathematically identical. The original LSTM uses multiplicative input and output gates [167]; later variants introduce a forget gate to regulate retention in the memory cell [117]. LSTMs proved far more effective at modeling long-range dependencies and became a pivotal advancement in neural language modeling [458, 128, 550]. Sutskever et al. [460] further demonstrated their scalability, showing that LSTMs could successfully model long sequences in large-scale neural machine translation using approximately 12 million sentence pairs. However, the inherently sequential nature of recurrent models restricts parallelization across time steps within a single training example, creating a substantial computational challenge for scaling to long sequences and large datasets [489].

A line of research attempted to further enhance recurrent architectures by improving their access to stored information. Attention mechanisms became widely adopted in neural sequence-to-sequence modeling beginning with the work of Bahdanau et al. [17], which allowed decoders to softly align to different parts of the source sequence at each decoding step. This content-based access significantly improved the model’s ability to handle long-term dependencies. Subsequently, attention rapidly became ubiquitous, applied to speech recognition [59], reading comprehension [53], sentence embedding [266], textual entailment [346], and summarization [351]. As Vaswani et al. [489] note, by the mid-2010s, attention mechanisms were an integral part of sequence modeling; however, they were nearly always used in conjunction with RNNs and thus did not remove the sequential bottleneck inherent to recurrent computation. Convolutional sequence models such as ByteNet [215] and ConvS2S [110] offered one route to parallel training by replacing recurrence with stacked (dilated) convolutions, but the number of layers needed to relate two distant positions still grows with their distance.

A different line of work pursued architectures that could replace recurrence altogether, which was started as early as 1991. Fast Weight Programmers (FWPs) [406, 408, 416] introduce the idea of a slow neural network that dynamically programs the fast weights [490, 97, 163] of other NNs. This explicitly separates storage and control [406], similar to traditional computers, allowing the network to rapidly encode short-term information without compressing it into static parameters.

A particularly relevant instance, which is now recognized as the Unnormalized Linear Transformer (ULTRA) [406, 408], implements an additive outer-product rule over key and value representations, offering a parallelizable formulation of attention (see Sec. 6.5). In retrospect, FWPs can be viewed as an early architectural direction where dynamic, rapidly updated weights provide associative memory, echoing principles that later proved central in modern attention mechanisms. As noted in 1993 [412] (where the terminology of learning “internal spotlights of attention” was introduced), “intra-sequence weight changes essentially occur only if the network pays a lot of attention to certain units by strongly exciting them or strongly inhibiting them”, hinting at the same context-dependent modulation in attention today. Later work revisited dynamic parameter generation, including HyperNetworks with recurrent applications [143] and analyses of attention as a hypernetwork [423].

The architectural developments led to the modern quadratic Transformer architecture [489], which, like ULTRA, completely removes recurrence and uses attention as its sole sequence-mixing primitive. Similarly, in the causal dot-product attention mechanism of modern Transformers, keys and values are generated by learnable model weights. For each output token, queries attend to the accumulated key–value collection through softmax-normalized aggregation to retrieve relevant information. This architectural shift enables full parallelization across sequence positions and has unlocked the massive scaling that defines modern large language models. The Transformer also became the vehicle of the pretrain-then-adapt paradigm: contextual representations from pretrained bidirectional LSTMs (ELMo [357]) were quickly superseded by pretrained Transformers, with GPT [378] and BERT [80] establishing decoder-only generative and encoder-only bidirectional pre-training, respectively, and the decoder-only recipe then scaled into the GPT-2 and GPT-3 lineage [379, 34] that defines today’s LLMs. As discussed in the introduction, recent architectural advances largely build upon refinements to the 2017 Transformer’s internal components, which we examine in detail in the subsequent sections.

4 Basic Modules in Transformer

In this section, we introduce the basic modules of the original *quadratic* Transformer³ architecture. The initial configuration [489] has a sequence-to-sequence framework that incorporates both an encoder and a decoder. Subsequent developments, as we will see in Sec. 4.7, include encoder-only, decoder-only (causal Transformers), and encoder-decoder architectures, which vary in their learning strategies. In general, the Transformer is characterized by its central components: the Transformer block, which consists of an attention module, a feed-forward network, residual connections, and normalization. Additionally, it features positional encoding accompanying the attention module, as well as the basic tokenizer and embeddings module.

The input and output structure of a Transformer model is as follows. The input sequence is first tokenized into a sequence of tokens using a tokenizer. These tokens are then mapped to initial embeddings through an embedding module. The initial embeddings are subsequently processed through L layers of Transformer blocks, producing the final contextual embeddings. Typically, these contextual embeddings are passed through a linear layer to generate the model’s output, the interpretation of which depends on the specific learning strategy employed.

4.1 Multi-head Attention (MHA)

The core computational primitive of the Transformer is the multi-head attention mechanism, which enables each token to dynamically attend to other tokens in the sequence through a Query-Key-Value (QKV) formulation. This design allows the model to capture long-range dependencies and diverse relational patterns within a single layer. Let $\mathbf{X} \in \mathbb{R}^{T \times d}$ denote the input to an attention layer, where T is the sequence length and d is the model dimensionality. Multi-head attention projects $\mathbf{X}$ into H parallel attention subspaces (heads), each operating on a lower-dimensional representation. For the h -th head, the query, key, and value matrices are obtained via learned linear projections:

$$\mathbf{Q}^{(h)} = \mathbf{X}\mathbf{W}_q^{(h)}, \quad \mathbf{K}^{(h)} = \mathbf{X}\mathbf{W}_k^{(h)}, \quad \mathbf{V}^{(h)} = \mathbf{X}\mathbf{W}_v^{(h)}, \quad (1)$$

where $\mathbf{W}_q^{(h)}, \mathbf{W}_k^{(h)}, \mathbf{W}_v^{(h)} \in \mathbb{R}^{d \times d_H}$ and $d_H = d/H$ denotes the dimensionality of each head.

³For simplicity, unless otherwise specified, we will use “Transformer” to refer to this architecture throughout the rest of this article.

Within each head, attention weights are computed by comparing queries and keys through scaled dot products. Specifically, the attention matrix for head h is defined as

$$\mathbf{A}^{(h)} := \text{softmax}\left(\frac{\mathbf{Q}^{(h)}\mathbf{K}^{(h)\top}}{\sqrt{d_H}}\right), \quad (2)$$

where the softmax is applied row-wise. The scaling factor $\sqrt{d_H}$ stabilizes optimization by preventing the dot-product magnitudes from growing excessively with dimension.

The output of the multi-head attention layer is obtained by applying the attention weights to the values within each head and then projecting the aggregated result back to the model dimension. A common formulation first concatenates the per-head outputs and applies a single output projection:

$$\text{Attn}(\mathbf{X}) = \text{Concat}\left(\mathbf{A}^{(1)}\mathbf{V}^{(1)}, \dots, \mathbf{A}^{(H)}\mathbf{V}^{(H)}\right) \mathbf{W}_o,$$

where $\mathbf{W}_o \in \mathbb{R}^{d_H \times d}$. Equivalently, it can be formulated as the “sum with per-head” form:

$$\text{Attn}(\mathbf{X}) = \sum_{h=1}^H \mathbf{A}^{(h)}\mathbf{V}^{(h)}\mathbf{W}_o^{(h)}, \quad (3)$$

where $\mathbf{W}_o^{(h)} \in \mathbb{R}^{d_H \times d}$ is the output projection matrix for head h . The equivalence follows by partitioning $\mathbf{W}_o \in \mathbb{R}^{Hd_H \times d}$ into H block rows, $\mathbf{W}_o = [\mathbf{W}_o^{(1)}; \dots; \mathbf{W}_o^{(H)}]$, so that multiplying the concatenated head outputs by $\mathbf{W}_o$ is exactly the sum of the head-wise products in Eq. (3). This formulation makes each head’s contribution explicit, which is convenient when analyzing head-level modifications (e.g., head reduction in Sec. 5.2).

Causal Attention in the Decoder. In decoder-only and autoregressive Transformer architectures, the attention mechanism is modified to enforce causality, ensuring that each token can attend only to itself and preceding tokens. This is achieved by adding a causal mask $\mathbf{M} \in \mathbb{R}^{T \times T}$ to the attention logits:

$$\text{CausalAttn}(\mathbf{X}) = \sum_{h=1}^H \text{softmax}\left(\frac{\mathbf{Q}^{(h)}\mathbf{K}^{(h)\top}}{\sqrt{d_H}} + \mathbf{M}\right) \mathbf{V}^{(h)}\mathbf{W}_o^{(h)}, \quad (4)$$

where

$$\mathbf{M}_{t,i} = \begin{cases} 0, & i \leq t, \\ -\infty, & i > t. \end{cases} \quad (5)$$

This lower-triangular mask prevents information leakage from future positions during training and inference.

Equivalently, for a given time step t , the causal attention output can be written in token-wise form as

$$\text{CausalAttn}(\mathbf{X})_t = \sum_{h=1}^H \left(\sum_{i \leq t} \alpha_{ti}^{(h)} \mathbf{v}_i^{(h)} \right) \mathbf{W}_o^{(h)}, \quad (6)$$

with attention weights

$$\alpha_{ti}^{(h)} = \frac{\exp\left(\mathbf{q}_t^{(h)} \mathbf{k}_i^{(h)\top} / \sqrt{d_H}\right)}{\sum_{j \leq t} \exp\left(\mathbf{q}_t^{(h)} \mathbf{k}_j^{(h)\top} / \sqrt{d_H}\right)},$$

where $\mathbf{q}_t^{(h)} = \mathbf{Q}_{t,\cdot}^{(h)}$, $\mathbf{k}_i^{(h)} = \mathbf{K}_{i,\cdot}^{(h)}$, and $\mathbf{v}_i^{(h)} = \mathbf{V}_{i,\cdot}^{(h)}$.

4.2 Feed-Forward Network

Complementing the attention mechanism, each Transformer block contains a position-wise Feed-Forward Network (FFN), which applies non-linear transformations independently to each token representation.

Formally, given an input matrix $\mathbf{X} \in \mathbb{R}^{T \times d}$, the FFN operates identically on every row and is defined as

$$\text{FFN}(\mathbf{X}) = \sigma(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \quad (7)$$

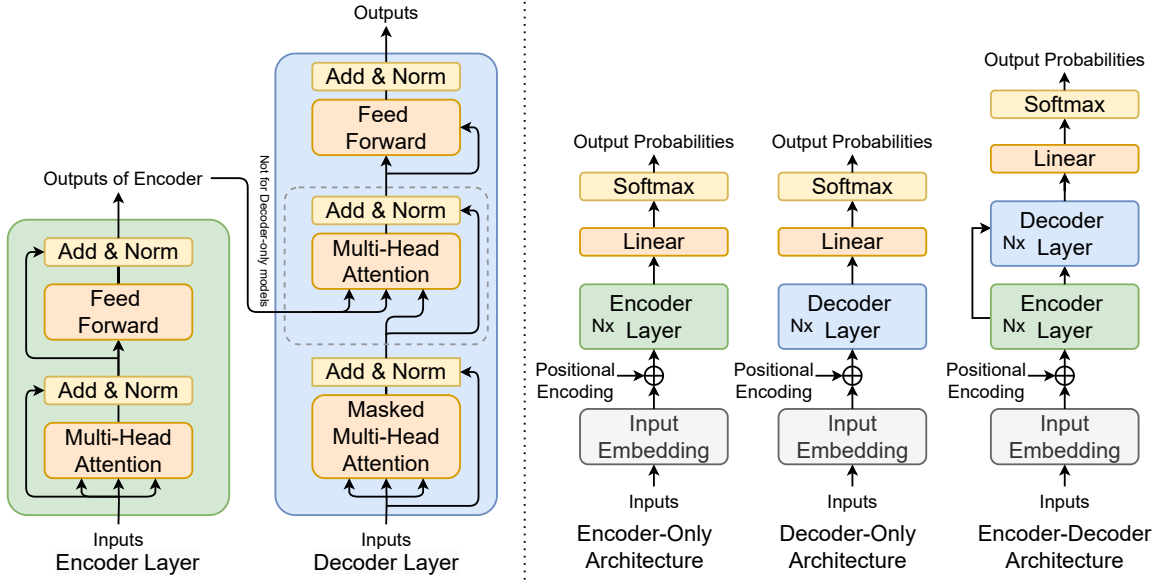


Figure 5: Structure of the Transformer layer and the encoder-decoder, encoder-only, and decoder-only architectures.

where $\sigma(\cdot)$ is a non-linear activation function. Here, $\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}$ and $\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}$ are learned weight matrices, and $\mathbf{b}_1, \mathbf{b}_2$ are bias vectors.

The intermediate dimensionality d_{ff} is typically chosen to be substantially larger than d (commonly $d_{ff} = 4d$), enabling the FFN to project token representations into a higher-dimensional space before mapping them back to the model dimension. With a conventional two-layer FFN and $d_{ff} \approx 4d$, the FFN contributes approximately two-thirds of the weight parameters within a standard Transformer block, excluding embeddings, the output vocabulary projection, and small bias or normalization terms (cf. Table 1). Its share of computation also depends on sequence length and execution phase: the attention interaction cost grows quadratically during a full-sequence forward pass, whereas the FFN cost grows linearly. This parameter concentration has motivated extensive research into FFN variants, including sparsification strategies such as Mixture of Experts (Sec. 7).

4.3 Positional Encoding

Unlike RNNs, which encode sequential order through their recurrence, Transformers can process all positions in parallel during training. Full self-attention without positional or other order-dependent signals is permutation-equivariant, so positional encodings provide explicit order information. Causal masking itself introduces ordering constraints, however, and models can recover aspects of position without explicit PE [188, 184].

Vaswani et al. [489] introduced sinusoidal positional encodings, which constitute a canonical form of absolute positional encoding (APE). More generally, APEs and alternative positional encoding schemes will be discussed in detail in Sec. 10.

Formally, let i denote the token position and let d be the model dimensionality (assumed to be even). The positional encoding vector $\mathbf{e}_i^{\text{pos}} \in \mathbb{R}^d$ is defined element-wise as

$$\mathbf{e}_{i,2j}^{\text{pos}} = \sin\left(\frac{i}{10000^{2j/d}}\right), \quad \mathbf{e}_{i,2j+1}^{\text{pos}} = \cos\left(\frac{i}{10000^{2j/d}}\right), \quad (8)$$

for $j = 0, \dots, \frac{d}{2} - 1$. Stacking these vectors for all positions yields the positional encoding matrix

$$\mathbf{E}^{\text{pos}} = [\mathbf{e}_1^{\text{pos}}; \dots; \mathbf{e}_T^{\text{pos}}] \in \mathbb{R}^{T \times d}. \quad (9)$$

In the original Transformer architecture, positional information is injected by adding $\mathbf{E}^{\text{pos}}$ to the token embeddings at the input layer, allowing the model to jointly reason over content and position. While this additive approach is widely

used, modern variants may inject positional information differently, such as within the attention mechanism itself, as discussed in Sec. 10.

4.4 Residual Connections and Normalization

In a Transformer block, the outputs of the MHA layer and the FFN layer are each followed by a residual connection and normalization. The original 2017 Transformer [489] employed *Post-LN*, where normalization (commonly Layer Normalization [13]) is applied after the residual connection:

$$\begin{aligned}\mathbf{X}' &= \text{Norm}(\text{Attn}(\mathbf{X}) + \mathbf{X}), \\ \mathbf{X}'' &= \text{Norm}(\text{FFN}(\mathbf{X}') + \mathbf{X}'),\end{aligned}\tag{10}$$

where $\text{Norm}(\cdot)$ denotes a normalization operator.

Together, residual connections and normalization support information propagation across layers and facilitate optimization of deep Transformer stacks; the Post-LN arrangement described here nevertheless becomes difficult to optimize at large depth without learning-rate warmup or careful initialization. Alternative arrangements of residual connections and normalization (e.g., Pre-LN) and other normalization variants are discussed in detail in Sec. 9.

4.5 Tokenization and Input Embeddings

In Transformer models, tokenization and embeddings define the interface between raw textual input and the continuous vector representations processed by the network. Given an input text sequence, a tokenizer first maps the text into a sequence of discrete tokens drawn from a finite vocabulary $\mathcal{V}$.

Let $\{x_i\}_{i=1}^T$, with $x_i \in \{1, \dots, |\mathcal{V}|\}$, denote the resulting token sequence of length T . Each token is mapped to a continuous vector representation via a learned embedding matrix $\mathbf{W}_E \in \mathbb{R}^{|\mathcal{V}| \times d}$, producing

$$\mathbf{e}_i = \mathbf{W}_E[x_i] \in \mathbb{R}^d,\tag{11}$$

where $\mathbf{W}_E[x_i]$ denotes the row of $\mathbf{W}_E$ indexed by token x_i . Stacking all token embeddings yields the embedding matrix $\mathbf{E} = [\mathbf{e}_1; \dots; \mathbf{e}_T] \in \mathbb{R}^{T \times d}$. In the original implementation [489], the input embeddings are scaled by $\sqrt{d}$.

The token embeddings are subsequently combined with positional encodings (cf. Sec. 4.3) to form the input to the first Transformer layer:

$$\mathbf{X}_0 = \mathbf{E} + \mathbf{E}^{\text{pos}}.\tag{12}$$

Thus, tokenization determines the discrete units over which the Transformer operates, while the embedding layer provides their initial continuous representations for subsequent computation.

4.6 Output Projection and Autoregressive Generation

The final hidden states produced by the Transformer are mapped back to the vocabulary space through an output projection. Given a hidden representation $\mathbf{h}_i \in \mathbb{R}^d$ at position i , the output projection maps it to vocabulary logits,

$$\mathbf{z}_i = \mathbf{h}_i \mathbf{W}_U + \mathbf{b},\tag{13}$$

where $\mathbf{W}_U \in \mathbb{R}^{d \times |\mathcal{V}|}$ and $\mathbf{b} \in \mathbb{R}^{|\mathcal{V}|}$. Applying a softmax converts these logits into a categorical distribution over the vocabulary:

$$\mathbf{p}_i = \text{softmax}(\mathbf{z}_i).\tag{14}$$

The output projection is often weight-tied with the input embedding matrix, such that $\mathbf{W}_U = \mathbf{W}_E^\top$.

Under the standard next-token prediction (NTP) formulation, the joint probability of a sequence is factorized into next-token conditional probabilities,

$$p(x_{1:T}) = \prod_{i=1}^T p(x_i \mid x_{<i}).\tag{15}$$

Accordingly, the distribution $\mathbf{p}_i$ produced at position i is trained to approximate the conditional distribution $p(\cdot \mid x_{\leq i})$, with the next token x_{i+1} as the prediction target. The NTP objective is then written as

$$\mathcal{L}_{\text{NTP}} = - \sum_{i=1}^{T-1} \log \mathbf{p}_i[x_{i+1}].\tag{16}$$

During autoregressive generation, a token is selected from the predicted distribution at each decoding step and appended to the current sequence, after which the model proceeds to predict the next token. Thus, the standard decoder generates sequentially while committing to one output token at each step.

4.7 Encoder/Decoder Transformer Configurations

Widely used Transformer-based models in natural language processing are commonly instantiated by choosing how to deploy the encoder and decoder components of the original sequence-to-sequence architecture. Concretely, retaining only the encoder yields *encoder-only* models, retaining only the decoder yields *decoder-only* (autoregressive) models, and combining both yields the full *encoder-decoder* configuration. Although these configurations share the same core building blocks as we introduced above, their information flow and training objectives differ, leading to distinct strengths for contextual representation learning, autoregressive generation, and sequence-to-sequence transformation, respectively.

4.7.1 Decoder-only

Decoder-only Transformers consist solely of a stack of Transformer decoder blocks and employ causal (masked) self-attention to model sequences autoregressively. At each position, the representation is computed using information from the current and preceding tokens, inducing a left-to-right factorization of the sequence probability. This design is particularly well suited for language modeling. The GPT (Generative Pre-trained Transformer) family [378, 34] is a representative instantiation of this architecture. In GPT-style models, each layer contains masked multi-head self-attention followed by a position-wise feed-forward network, without the encoder–decoder cross-attention present in the original Transformer decoder (Figure 5). By scaling depth, width, and training data, decoder-only Transformers have become the dominant paradigm for large-scale generative language models.

4.7.2 Encoder-only

Encoder-only Transformers retain only the encoder stack of the original architecture and leverage unrestricted self-attention, allowing each token to attend to all other tokens in the sequence (Figure 5). BERT (Bidirectional Encoder Representations from Transformers) [80] exemplifies this configuration. Through bidirectional pre-training objectives, encoder-only models learn contextualized token representations that capture semantic and syntactic relationships across the entire input sequence, which can be readily adapted to downstream tasks such as classification, retrieval, and span prediction.

4.7.3 Encoder–Decoder

Encoder–decoder Transformers preserve the full structure of the original sequence-to-sequence architecture. In this configuration, the encoder maps the input sequence into a set of contextual representations, while the decoder generates the output sequence autoregressively, attending both to previously generated tokens and to the encoder outputs via cross-attention. This separation between encoding and generation provides a natural inductive bias for sequence transformation tasks. The T5 (Text-to-Text Transfer Transformer) model [381] is a prominent example of this architecture. T5 adopts a unified text-to-text formulation, in which both inputs and outputs are represented as token sequences, allowing the same encoder–decoder framework to be applied across tasks such as translation, summarization, and question answering.

Despite their natural fit for sequence-to-sequence transformation, encoder–decoder Transformers are comparatively less common among contemporary general-purpose LLMs. The reasons are less about asymptotic cost (an encoder pass is comparable to a decoder-only prefill, and cross-attention to encoder states is comparable to attending over a cached prefix) than about simplicity and generality: decoder-only models condition on inputs by simple prefixing within a single autoregressive stack, treat prompts, generated text, and multi-turn continuations uniformly, and were found to generalize better zero-shot after pure language-model pre-training [502]. This offers a more streamlined architecture for instruction following and open-ended generation.

4.8 Computational Complexity

The computational cost of the Transformer architecture is primarily driven by its two main components: the multi-head attention mechanism and the feed-forward networks. Figure 6 provides a visualization of the key matrix operations

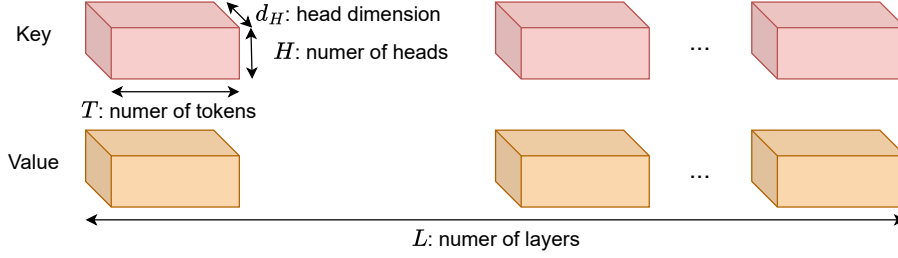


Figure 6: A visualization of the key matrix operations and their dimensions within the multi-head attention mechanism.

	Parameters	Training Complexity	Decoding Complexity	KV cache
multi-head attention	$4d^2L$	$O(T^2dL + Td^2L)$	$O(TdL + d^2L)$	$2TdL$
feed-forward network	$8d^2L$	$O(Td^2L)$	$O(d^2L)$	-

Table 1: Computational and memory requirements of Transformer components. Training complexity is the cost of one forward pass over a sequence of length T ; decoding complexity is the cost per generated token when the context length is T (with KV cache). For attention, the Td^2L (resp. d^2L) term is the cost of the Q/K/V/output projections and dominates whenever $T < d$.

within the MHA. Assuming a typical configuration where the intermediate dimension of the FFN is $d_{\text{ff}} = 4d$, the computational and memory requirements are summarized in Table 1. Note that the model dimension d is composed of H attention heads, each with a dimension of d_H , such that $d = H \cdot d_H$.

As shown in the table, the two columns measure different things: the training column is the cost of processing an entire sequence in parallel, whereas the decoding column is the cost of producing a single additional token given a context of length T . Making the per-token decoding cost linear in T relies on the *KV cache* [432, 189, 147], a fundamental optimization technique that has been widely used in Transformer implementations.

To understand the necessity of KV cache, consider the autoregressive generation process in decoder models. When generating text, each new token must attend to all previously generated tokens in the sequence. From Eq. (6), we observe that computing the output for token t requires the Key and Value vectors $\mathbf{k}_i^{(h)}$ and $\mathbf{v}_i^{(h)}$ for all positions $i \leq t$. Naively, at each generation step, we would need to recompute these K and V vectors for the entire preceding context, which is highly redundant since these vectors remain unchanged once computed.

The KV cache addresses this inefficiency by storing the computed Key and Value states from previous time steps. When generating token $t + 1$, instead of recomputing $\mathbf{k}_i^{(h)}$ and $\mathbf{v}_i^{(h)}$ for all $i \leq t$, the model simply retrieves them from the cache and only computes the K and V vectors for the new token $t + 1$. This cached approach fundamentally alters the computational profile of the attention mechanism, leading to the following two distinct operational modes:

- **Parallel Processing (Training & Pre-filling):** In this mode, the model processes the entire input sequence in a single forward pass. This occurs during training for efficiency and during the inference pre-fill stage, where the initial prompt is encoded to populate the KV cache. In both cases, the full attention matrix is computed, leading to an attention cost of $O(T^2dL)$ (in addition to the $O(Td^2L)$ projections), which scales quadratically with the sequence length (T).
- **Sequential Generation (Decoding):** In this mode, the model generates the output autoregressively, i.e., one token at a time. This phase relies heavily on the KV cache. At each step, the model takes the newest token, computes its Query (Q) vector, and retrieves the cached Key (K) and Value (V) vectors from all previous steps. By only attending to the cached K/V pairs, the model avoids recomputing these values for the entire context. Without the cache, every step would have to recompute the key and value projections of all T context tokens, costing $O(Td^2L)$ per step in projections alone; with the cache, the per-step cost is $O(TdL + d^2L)$, of which the $O(TdL)$ term is the attention over the cached context and the $O(d^2L)$ term is the projections of the single new token.

5 Efficient Attention and KV Cache Management

Despite the remarkable success of Transformer-based large language models in language processing tasks [338], they encounter two distinct efficiency bottlenecks when processing and generating long sequences. During training and prefill, the dot-product attention mechanism incurs a computational cost that grows quadratically with the input length. During autoregressive decoding, the per-step attention cost is linear in the context length, but the key–value (KV) cache that must be stored and re-read at every step grows linearly with the context, so memory footprint and memory bandwidth become the dominant constraints.

As introduced in Sec. 4.8, the size of the KV cache, and hence the memory traffic of decoding, is governed by four axes: sequence length T , the number of key–value heads H_{kv} , the per-token size d_H of each cached key or value, and the number of layers L . For conventional KV heads, the cache contains $2LTH_{kv}d_H$ elements. We distinguish H_{kv} from the query-head count H_q , with $d = H_qd_H$ under the standard head-width convention. Most current methods specialize in a single axis: token-oriented approaches reduce the number of cached or attended positions (e.g., SepLLM [46]); head-oriented approaches reduce the number of KV heads shared among query heads (e.g., GQA [1]); and dimension-oriented approaches compress the per-token KV representation into a low-rank latent shared across heads (e.g., MLA [73]). The fourth axis, the number of layers L , is exploited by layer-oriented approaches that share or merge KV states across adjacent layers (e.g., MiniCache [270]). We note that these axes primarily target KV cache memory and decoding-time memory traffic; only a subset of token-oriented methods, mainly those that sparsify attention during prefill, also reduce the quadratic attention computation itself, and head- or dimension-oriented compression can even increase attention FLOPs while shrinking the cache.

5.1 Token Reduction

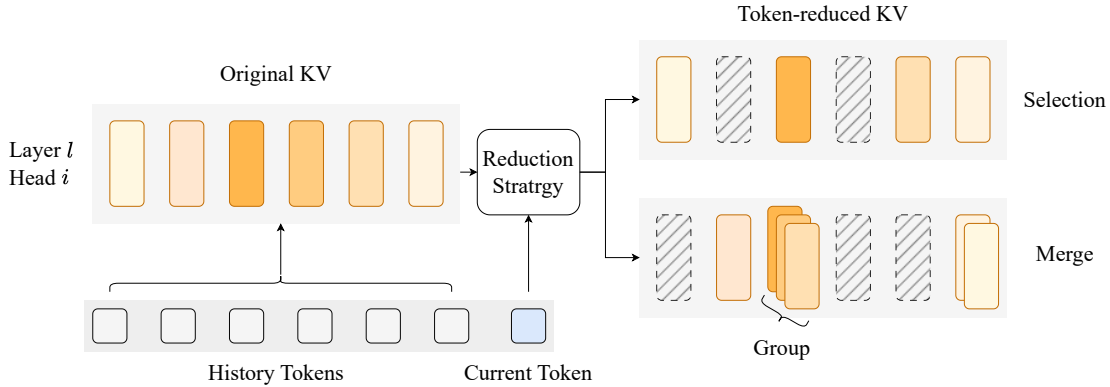


Figure 7: Token reduction strategies towards KV in attention mechanism.

Recently, a broad class of *token reduction* techniques has been proposed to improve the efficiency of attention by reducing the number of stored or attended entries through selection or compression. These methods primarily aim to alleviate KV cache memory pressure and decoding-time memory traffic, thereby improving inference throughput, often with limited degradation in model quality; a subset of them, mainly prefill-oriented or training-based sparse-attention designs, also reduces the quadratic cost of attention itself, whereas methods that compress the cache after a dense prefill do not. Representative approaches are illustrated in Figure 7 and summarized in Table 2, which groups them by the mechanism through which attention cost is reduced: (a) eviction and compression methods that bound the cache, so that unselected entries are gone; (b) methods that retain the full cache and sparsify only which entries each decoding step reads, i.e., inference-time approximations of dense attention; and (c) natively sparse attention, in which the sparsity pattern is produced by a trained component and is part of the model definition. The remaining columns record design dimensions that cut across the three groups: when the selection is made (Phase), what training it requires (Train.), which cost it reduces (Saves), which tokens are exempt from selection (Kept), the signal used to score tokens, whether unselected entries are dropped, merged, or kept, and the shape (Budget) and unit (Gran.) of the selection. We discuss the three groups in turn and then these dimensions.

At a high level, token reduction methods define an importance or relevance metric over tokens and apply a selection rule (e.g., top- k or thresholding) to determine which keys and values should remain active. For selection-based methods, an additional *selection mask* sparsifies attention by disallowing interactions with unselected entries. This mask describes which entries are read or retained; merging and compression instead construct new KV representations through an aggregation or encoding map and are not captured by a binary mask alone.

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_H}} + \mathbf{M} + \mathbf{M}_{\text{sel}}\right), \quad (17)$$

where $\mathbf{M}$ is the standard causal mask, and $\mathbf{M}_{\text{sel}} \in \mathbb{R}^{T \times T}$ is defined as

$$(\mathbf{M}_{\text{sel}})_{i,j} = \begin{cases} 0, & \text{if token } j \text{ is retained (selected) for query position } i, \\ -\infty, & \text{otherwise.} \end{cases} \quad (18)$$

Which of the three costs a method saves follows from how $\mathbf{M}_{\text{sel}}$ is realized: if pruned entries are removed from the cache, or never written to it, memory is saved; if the mask is realized by a kernel that skips the unselected pairs during prefill or training, rather than by masking a fully materialized score matrix, the attention compute falls from $\mathcal{O}(T^2)$ to the number of selected pairs, although a selector that scores every pair, such as an indexer, keeps a quadratic term of its own; if the mask only restricts the entries read at each decoding step, memory bandwidth is saved.

(a) KV Eviction and Compression. Methods in group (a) of Table 2 bound the cache by discarding or merging entries. Together with MoSA in group (c), they are the only methods that reduce KV memory without moving it elsewhere, and the only ones in which a wrong decision is irreversible; their evolution is largely a search for better proxies of a token’s future usefulness.

The earliest proxies use position alone. StreamingLLM [519] observes that autoregressive LLMs place a large share of attention on the first few tokens regardless of their content, an *attention sink*, and that window attention collapses as soon as these tokens are evicted; keeping the sinks plus a rolling window of recent tokens, with positions re-indexed relative to the cache, gives stable perplexity on streams of millions of tokens without any training. LM-Infinite [145] arrives at the same Λ -shaped pattern from a length-generalization analysis, adds a ceiling on the relative distances fed to the positional encoding, and optionally lets each head in the upper layers attend to the top- k middle tokens by attention logit.

Content-based eviction during decoding replaces position with attention statistics. Scissorhands [277] formulates the *persistence of importance* hypothesis, that tokens receiving high attention at one step keep receiving it later, verifies it empirically, and evicts under a fixed budget using importance accumulated over a history window while always keeping the most recent tokens. H₂O [570] keeps the *heavy hitters*, the few tokens that accumulate most of the attention mass, alongside an equal budget of recent tokens, and shows that this greedy policy comes with a guarantee under a dynamic submodular view of the cache. BUZZ [572] refines the heavy-hitter idea by segmenting the history outside the sinks and the sliding window into chunks and retaining the strongest tokens of each chunk, sampling new and old tokens at different strides, in both prefill and decoding. FastGen [109] instead profiles each head once, during prompt encoding, and assigns it one of several caching policies: heads that attend locally drop distant context, heads that attend to special tokens or punctuation drop everything else, and only broadly attending heads keep the full cache.

A second lineage compresses the prompt once, after a dense prefill. SnapKV [255] observes that the attention pattern of the last few prompt tokens, the *observation window*, predicts what will be attended during generation, and per head keeps the window plus the prompt positions it attends to most, after pooling neighbouring scores so that selected positions form clusters. Its selection rule became the substrate for a family of budget schemes. PyramidKV [566] decreases per-layer budgets arithmetically with depth, following the funneling of attention toward a few tokens in upper layers; DynamicKV [578] sets per-layer budgets from the task-specific attention observed during prefill and updates the preceding layers progressively; PrefixKV [494] searches a global prefix configuration that maximizes the cumulative priority retained in each layer, for vision instruction-following models; CAKE [367] derives layer preferences from the spatial dispersion and temporal shift of attention, manages the resulting budgets in a cascading manner within a peak-memory limit during prefill, and adds an eviction indicator that accounts for the variability of a token’s importance over time; AdaKV [98] bounds the attention-output loss of eviction and shows that a global top- k across the heads of a layer, which gives concentrated heads small budgets and dispersed heads large ones, tightens

Method	Year	Phase	Train.	Saves	Kept	Scoring signal	Cache	Budget	Gran.
<i>(a) KV eviction and compression: the cache is bounded</i>									
Scissorhands [277]	2023	D	–	M+B	R	attention over a history window	Drop	U	T
H ₂ O [570]	2023	D	–	M+B	R	accumulated attention	Drop	U	T
LM-Infinite [145]	2023	P+D	–	M+B	S+R	position (opt. top- k middle tokens)	Drop	U	T
StreamingLLM [519]	2023	P+D	–	M+B	S+R	position only	Drop	U	T
FastGen [109]	2023	P+D	–	M+B	S+R	per-head attention profile	Drop	H	T
DMC [325]	2024	P+D	Retro	M+B	–	learned append/merge decision	Merge	H	T
SnapKV [255]	2024	P	–	M+B	R	obs.-window attention (pooled)	Drop	U	T
PyramidKV [566]	2024	P	–	M+B	R	obs.-window attention	Drop	L	T
AdaKV [98]	2024	P	–	M+B	R	obs.-window attention, global top- k	Drop	H	T
RazorAttention [461]	2024	P+D	–	M+B	S+R	retrieval-head test; comp. token	Drop	H	H
NACL [49]	2024	P	–	M+B	R	proxy-token attention + random	Drop	U	T
DuoAttention [520]	2024	P+D	Aux	M+B	S+R	learned retrieval-head gates	Drop	H	H
SimLayerKV [564]	2024	P+D	–	M+B	S+R	lazy-layer detection	Drop	L	L
HeadKV [104]	2024	P	–	M+B	R	head retrieval–reasoning score	Drop	H	T
BUZZ [572]	2024	P+D	–	M+B	S+R	segment heavy hitters + sampling	Drop	U	T
PrefixKV [494]	2024	P	–	M+B	–	layer-wise cumulative priority	Drop	L	T
LeanKV [565]	2024	P+D	–	M+B	–	attention tiers + mixed precision	Drop	H	T
SepLLM [46]	2024	P+D	Scr./Retro	M+C+B	S+R	separator tokens (rule)	Drop	U	T
DynamicKV [578]	2024	P	–	M+B	R	recent-query attention	Drop	L	T
CriticalKV [99]	2025	P	–	M+B	R	output perturbation (attn. $\times$ value)	Drop	U/H	T
CAKE [367]	2025	P	–	M+B	R	attention dispersion + temporal shift	Drop	L	T
KVzip [223]	2025	P	–	M+B	–	context-reconstruction attention	Drop	H	T
DropKV [554]	2026	P	–	M+B	R	decoupled output perturbation	Drop	U	T
ReasonAlloc [274]	2026	D	–	M+B	R	plug-in layer/head budget routing	Drop	L+H	T
KVpop [150]	2026	P+D	Retro	M+B	S+R	learned scorer (future attention)	Drop	U	T
BeaconKV [224]	2026	D	–	M+B	R	beacon + recent query attention	Drop	U	T
<i>(b) Sparse reads over a retained cache</i>									
SparQ [386]	2023	D	–	B	–	top- r query dims + mean value	Keep	U	T
InflLM [518]	2024	P+D	–	M+B	S+R	representative-token similarity	Off.	U	B
Quest [462]	2024	D	–	B	–	page min/max key bounds	Keep	U	B
QuickLLaMA [250]	2024	P+D	–	M+B	S+R	query-aware block lookup	Off.	U	B
SampleAttention [586]	2024	P	–	C	S+R	sampled column stripes + window	Keep	H	T
InfiniGen [244]	2024	D	–	M+B	–	next-layer rehearsal attention	Off.	U	T
PQCache [556]	2024	D	–	M+B	S+R	PQ-approximated $q \cdot k$	Off.	U	T
EM-LLM [101]	2024	P+D	–	M+B	S+R	surprise events + k -NN similarity	Off.	U	E
RetrievalAttention [272]	2024	D	–	M+B	S+R	attention-aware ANNS	Off.	U	T
MagicPIG [52]	2024	D	–	M+B	S+R	LSH importance sampling	Off.	U	T
SqueezedAttention [169]	2024	P+D	–	C+B	–	hierarchical centroid lookup	Keep	H	C
RefreshKV [525]	2024	D	–	B	–	periodic full-attention refresh	Keep	U	T
AttentionPredictor [536]	2025	D	Aux	M+B	S+R	CNN-predicted attention	Off.	U	B
<i>(c) Natively sparse attention: the sparsity pattern is learned as part of the model</i>									
MoBA [282]	2025	P+D	Scr./Retro	C+B	R	query $\cdot$ mean-pooled block key	Keep	U	B
NSA [547]	2025	P+D	Scr.	C+B	R	compression-branch attention	Keep	U	B
MoSA [359]	2025	P+D	Scr.	M+C	–	expert-choice router (non-causal)	Drop	U	T
DSA [75]	2025	P+D	Retro	C+B	–	lightning indexer (KL-aligned)	Keep	U	T
MSA [311]	2026	P+D	Scr.	C+B	R	index branch, per GQA group	Keep	U	B
HiLS-Attention [174]	2026	P+D	Scr./Retro	C+B	R	landmark chunk scores (LM loss)	Keep	U	B

Table 2: Comparison of token reduction methods, grouped by mechanism and ordered by the year of the first public version. **Phase:** P = selection is decided once on the prompt after prefill; D = repeated during decoding; P+D = applied throughout the sequence. **Train.:** – = training-free; Aux = only an auxiliary module is trained and the LLM is frozen; Retro = an existing LLM is further trained; Scr. = trained from scratch. **Saves:** M = KV memory on the GPU (for offloading methods it is moved to host memory rather than discarded); C = attention compute in prefill or training; B = per-step KV reads (memory bandwidth) in decoding. **Kept:** S = initial (sink) tokens and R = a recent window (or the current block) are exempt from selection. **Cache:** Drop = unselected entries are discarded; Merge = merged into fewer entries; Off. = offloaded to host memory; Keep = the full cache stays on the GPU and only the reads are sparse. **Budget:** U = one budget shared by all layers and heads; L = layer-adaptive; H = head-adaptive; L+H = layer-wise then head-wise. **Gran.:** the unit of the keep or read decision, T = token, B = block or page, C = cluster, E = event, H = head, L = layer.

the bound; and HeadKV [104] sizes head budgets from an offline estimate of each head’s retrieval-and-reasoning importance. NACL [49] also evicts in a single operation during encoding but scores tokens by the attention they receive from *proxy tokens*, typically the question at the end of the prompt, and mixes in random eviction to counter the bias of attention statistics. Scoring criteria have since moved from attention weights to the perturbation of the attention output. CriticalKV [99] shows that the value states and the output projection also determine how much dropping an entry perturbs the output and derives a perturbation-constrained selection rule that plugs into existing methods; DropKV [554] notes that choosing the evicted set that minimizes this perturbation is NP-hard, decouples it into independent per-token scores that combine a token’s attention weight with the residual of its value vector relative to the attention output, and proves a constant-factor approximation guarantee under an empirically verified *cone condition*, with fused kernels that avoid materializing the attention matrix.

Coarser than any of these, a third lineage decides per head or per layer. RazorAttention [461] identifies a small fraction of *retrieval heads* (about 15% in its setting) through echo and induction tests, keeps their cache intact, reduces all other heads to sinks and a recent buffer, and folds each head’s dropped entries into a single compensation token; DuoAttention [520] learns the same split with one trainable gate per head, optimized on synthetic retrieval data with the LLM frozen, and streams the non-retrieval heads at inference; SimLayerKV [564] identifies *lazy layers*, whose attention concentrates on the initial and recent tokens, at prefill time and replaces their full attention with streaming attention. LeanKV [565] adds a precision axis, sorting each head’s tokens by attention into full-precision, quantized, and evicted tiers under per-head dynamic budgets.

Two recent directions relax the dependence on the queries seen so far. KVzip [223] scores each entry by how much the frozen model attends to it when prompted to reconstruct the context from the cache, so that the compressed cache can be reused across arbitrary later queries; it reports a 3–4× reduction at negligible loss across question answering, retrieval, reasoning, and code tasks. For long chain-of-thought decoding, where the cache is dominated by generated tokens, BeaconKV [224] shows that certain decoding steps re-attend to distant context, such as a plan formulated early in the trace, which the most recent queries do not anticipate; observing that the queries of such steps cluster into a small number of groups, it maintains a compact set of *beacon queries* that are refreshed during decoding and scores cached entries with the beacon queries together with the recent ones while always retaining a recent window, reporting up to 5.8× memory reduction on open-weight reasoning models while nearly preserving full-cache accuracy. ReasonAlloc [274] addresses the same regime at the budget level: it combines an offline layer-wise pre-allocation, following an architecture-dependent demand profile the authors call a “reasoning wave”, with an online head-wise reallocation toward information-rich heads during decoding, and is plug-and-play with existing eviction rules such as SnapKV.

Finally, the policy itself can be learned. DMC [325] retrofits a pretrained model, by continued pre-training on a negligible fraction of the original data and without new parameters, to predict for every new key–value pair whether to append it or to accumulate it into the most recent entry, yielding learned per-head compression ratios without any entry being discarded outright. KVpop [150] keeps sinks and a protected window of recent tokens per KV head and trains lightweight per-head scorers to rank the remaining tokens for a fixed long-range top- k budget; the scorers are supervised by the attention a token receives *after* it leaves the protected window, i.e., at the eviction boundary where the keep-or-drop decision is actually made, and the backbone is adapted to the resulting sparse pattern by distillation from the full-attention model. SepLLM [46] builds on the observation that separator tokens receive disproportionately large attention and compresses each segment into the KV of its separator, keeping separators, initial tokens, and neighbouring tokens; it can be applied training-free, by post-training, or from scratch.

Across the group the trade-off is uniform: memory falls in proportion to the budget, but the retained set is chosen against the queries available at eviction time. Methods that compress the prompt approach full-attention quality on single-query benchmarks at moderate budgets, lose accuracy under aggressive compression, and can degrade in multi-turn use because context relevant to a follow-up question may already be gone; the extent of this degradation depends on the method, budget, and task. Decoding-time methods face the analogous risk that a token discarded early in a reasoning trace is needed later.

(b) Sparse Reads over a Retained Cache. Group (b) starts from two observations: decoding is bound by memory traffic rather than arithmetic, and the entries a query needs are few but change from query to query. Rather than deciding once which entries to keep, these methods retain every entry, on the GPU or in host memory, and estimate cheaply at each step which ones to read. The model parameters are unchanged and no entry is permanently discarded; each step,

however, attends over an estimated subset, so its output is an approximation of dense attention whose error can be traded against the number of entries read.

The simplest estimators approximate the scores of individual tokens or pages. SparQ [386] uses only the r largest-magnitude components of the current query to approximate the attention scores from the corresponding slice of the key cache, fetches the full keys and values of the top- k tokens, and adds the mean of the remaining values to compensate for the mass it skips. Quest [462] works at page granularity: it stores per-dimension minima and maxima of the keys in each page, so that an upper bound on any query’s score for that page can be computed from the metadata alone, and loads only the top- k pages. InfiniGen [244] exploits the similarity of consecutive layers’ inputs to rehearse the next layer’s attention at the current layer, using skewed partial query and key weights prepared during prefill, and prefetches only the predicted-critical entries from a CPU-resident cache pool. AttentionPredictor [536] learns the estimator: a small convolutional model, trained offline on collected attention maps, predicts the next step’s attention from its recent temporal pattern at block granularity, recomputes dense attention periodically to correct drift, and drives block prefetching from an offloaded cache.

A second family organizes the cache as a retrievable memory. InfLLM [518] splits the distant context into block-level memory units, represents each unit by the few tokens that received the most attention within it, retrieves the units most relevant to the current tokens alongside the initial tokens and a local window, and offloads rarely used units to the CPU; because the local window bounds the effective attention span, it also lets models trained on short contexts process sequences of up to 1M tokens without training. QuickLLaMA [250] builds on it by keeping the user’s question in the window so that the lookup is query-aware. EM-LLM [101] segments the stream into variable-length *episodic events* at points of high Bayesian surprise, refines the boundaries with graph-theoretic criteria, and retrieves events by k -NN similarity together with their temporal neighbours; since only the events read at a step reside on the GPU, context length is limited by CPU memory or disk rather than by GPU memory. PQCache [556] treats attention as an embedding-retrieval problem and borrows the classic tools of approximate nearest-neighbour search (ANNS) [23]: during prefill it product-quantizes [202] the keys, clustering each subspace with K-means, and keeps the full cache in CPU memory; during decoding it approximates scores from the codes and centroids and fetches the top- k tokens, deliberately at token rather than block granularity because relevant tokens are scattered. RetrievalAttention [272] notes that queries and keys follow different distributions, so that off-the-shelf approximate nearest-neighbour indexes over keys must scan a large fraction of the vectors; it builds an attention-aware index that connects the prefill queries to their exact nearest keys, so that a decoding query first finds similar past queries and through them the relevant keys, keeps a static set of initial and recent tokens on the GPU, and merges the partial attention computed on the CPU and the GPU. MagicPIG [52] questions top- k retrieval itself: attention is often less sparse than its sink-dominated shape suggests (the top 20% of keys may cover only 70% of the mass), so it replaces selection with self-normalized importance sampling whose proposal distribution comes from locality-sensitive hashing, runs the hash tables and the sampled attention on the CPU, and keeps sinks and local tokens on the GPU. SqueezedAttention [169] targets a fixed context shared across requests: it clusters the keys offline into a hierarchy of centroids, compares each query with the centroids online to retrieve the relevant clusters, and calibrates the threshold so that heads with flatter attention read more keys, accelerating both prefill and generation.

Two members differ in when they act. SampleAttention [586] accelerates prefill rather than decoding: it samples a few query rows per chunk to find the head-specific column stripes and local window that carry a target fraction of cumulative attention, then computes only those blocks. RefreshKV [525] keeps a small cache of the tokens most attended at the last full-attention step and decides at every layer and step, from the similarity between the current query and that step’s query, whether to attend over the subset or to refresh it with a full pass.

The advantages of the group mirror the weaknesses of group (a): every query can reach the full history, so a follow-up question is not penalized by an eviction made for an earlier one, and quality is governed by the per-step read budget and the recall of the estimator rather than by a one-shot decision; a read that misses a relevant entry still changes the output, and that error can propagate into later hidden states. Its costs are that GPU memory is saved only by offloading, which makes PCIe bandwidth and CPU compute the new bottleneck; that indexes or metadata must be built at prefill; and that the estimators add per-step work of their own.

(c) Natively Sparse Attention. Group (c) makes sparsity part of the model: the pattern is produced by a trained component, so there is no approximation of a dense model, at the price of training. Whether the pattern is also applied identically in training, prefill, and decoding, and hence whether the train–inference discrepancy disappears, has to be

checked per method: a causal selector (MoBA, NSA, DSA, MSA, HiLS-Attention) can be used unchanged at decoding time, whereas a selector that ranks all tokens of a sequence cannot. MoBA [282] divides the preceding context into fixed-length blocks and, for each query, computes an affinity score against a mean-pooled key of every block; the query then attends only to its top- k blocks together with the current block. Unselected blocks are not evicted: they remain in the KV cache and may be selected by other queries or heads. NSA [547] uses a dynamic hierarchical sparse strategy that combines coarse-grained token compression, fine-grained block selection whose scores are derived from the compression branch, and a sliding-window branch, so as to maintain both global context awareness and local precision with hardware-aligned kernels. MoSA [359] selects, for each sparse head, the top- k tokens of the sequence with expert-choice routing and computes the query, key, and value transformations only for those tokens, which shrinks the key-value count of a sparse head from T to k ; a few dense heads are retained, so the total count is $TH_{\text{dense}} + kH_{\text{MoSA}}$. Because the top- k is taken over the whole sequence, the routing is non-causal: the original paper evaluates MoSA with teacher forcing and lists an autoregressive adaptation, e.g., a learned causal predictor as in Mixture-of-Depths, as future work, so its decoding-time behaviour is not yet established. We return to MoSA from the routing perspective in Sec. 7.1.

Recent advancements in million-token context large language models have increasingly relied on training-based sparse attention architectures to overcome the quadratic computational complexity of dense attention. Unlike training-free heuristic masks, these methods integrate dynamic, learnable sparsity directly into the pre-training or continued pre-training phases. DeepSeek Sparse Attention (DSA), introduced in DeepSeek-V3.2 [75], trains a lightweight *lightning indexer* that scores past tokens for each query and restricts attention to the top- k scored entries on top of an MLA backbone. Building on this, DeepSeek-V4 [76] interleaves Compressed Sparse Attention (CSA), which applies DSA-style indexer-based top- k selection over compressed key-value entries, with Heavily Compressed Attention (HCA), which attends densely over a much more aggressively compressed set of entries without an indexer. GLM-5 [122] adopts DSA through a cost-effective two-stage continued pre-training strategy, further stabilizing the indexer during long-horizon reinforcement learning via deterministic top- k operators. To address the memory-wall bottlenecks on specialized AI accelerators, LongCat-2.0 [291] proposes LongCat Sparse Attention (LSA). LSA incorporates streaming-aware indexing to enable hardware-friendly coalesced memory access and utilizes cross-layer distillation to amortize the indexer’s computational overhead across multiple Transformer layers.

MiniMax Sparse Attention (MSA) [311] instead builds block-sparse attention on top of GQA rather than MLA: a lightweight index branch, with one index query head per GQA group and a single index key head, scores key-value blocks by max-pooled similarity and selects a separate top- k set of blocks for each group (the local block is always included); the main branch then performs exact block-sparse attention over the selected blocks, and a KL loss aligns the index distribution with the group-averaged attention distribution of the main branch. On a 109B-parameter natively multimodal model, MSA is reported to match GQA quality while reducing per-token attention compute by 28.4× at a 1M context.

HiLS-Attention [174] targets a limitation shared by chunk-wise designs such as MoBA, NSA, and MSA: the chunk scores are used only to pick a hard top- k set and are then discarded, so the language-modeling loss cannot directly optimize the selection (DSA and MSA instead train their indexers with an auxiliary KL objective). It appends a landmark token to each chunk to obtain a learnable chunk summary and factorizes attention into an inter-chunk softmax over chunk summaries followed by an intra-chunk softmax within the selected chunks, so that the selection scores are trained end-to-end by the language-modeling loss; after continued pre-training of a 7B model on 50B tokens, it is reported to retain full-attention quality in-distribution and to extrapolate to more than 64× the training context length.

Once attention itself is sparse, the indexer that scores the whole prefix for every query becomes the remaining $O(T^2)$ term. Two training-free remedies for DSA-style models have been proposed: HISA [528] replaces the flat token scan with a block-level coarse filter followed by token-level refinement inside the retained blocks, producing the same top- k token pattern as the original indexer with up to 3.75× faster indexing at 64K context, and IndexCache [22] exploits the similarity of top- k selections across adjacent layers by running the indexer only in a subset of layers and reusing its indices in the others, removing up to 75% of indexer computation; both have been applied to DeepSeek-V3.2 and GLM-5 without retraining the attention weights.

Collectively, these reports indicate that learning the sparsity pattern jointly with the model weights, often combined with KV compression, can approach full-attention quality at a fraction of the attention cost; controlled comparisons against training-free methods under matched budgets remain limited, however.

Scoring Signal. The Scoring-signal column of Table 2 orders these proxies by how much of the future they try to anticipate. Position-only rules (StreamingLLM, LM-Infinite) and the separator rule of SepLLM use no content: they are trivially reusable across queries and cost nothing at selection time, but are task-agnostic. Attention accumulated over past queries (Scissorhands, H₂O) and attention from a recent observation window (the SnapKV family, CAKE, DropKV) assume that what recent queries attended to is what future queries will need. This assumption holds well within a single turn and is what makes prompt compression competitive at moderate budgets, but it fails across turns and in long reasoning traces, which is precisely the motivation for KVzip’s reconstruction-based scoring and BeaconKV’s beacon queries. Similarity to the current query, whether estimated by approximate nearest-neighbour search (RetrievalAttention, PQCache), bounded per page (Quest), or sampled (MagicPIG), is the most faithful signal, but it is available only when the full cache is retained, i.e., in group (b). Learned scorers can encode any of these cues; they are supervised by future attention (KVpop), by the deviation from the full-attention output (DuoAttention), by the language-modeling loss under a compression penalty (DMC), by the attention distribution of the main branch (DSA, MSA), or directly by the language-modeling loss through the factorized inter-/intra-chunk softmax of HiLS-Attention. Every signal except the position rules adds computation at selection time; in group (c) this indexer cost is what remains of the quadratic term, and HISA and IndexCache exist to reduce it.

Phase and Protected Tokens. The Phase column records when the selection is made. Prompt-compression methods such as SnapKV, PyramidKV, AdaKV, NACL, and KVzip decide once, on the prompt, after a dense prefill; the tokens generated afterwards are cached in full, so these methods bound the memory of long inputs but not of long outputs. Decoding-time methods such as H₂O and Scissorhands, and the retrieval methods of group (b), re-select at every decoding step, which is what long chain-of-thought generation requires; BeaconKV and ReasonAlloc are designed specifically for this regime, and SampleAttention, which sparsifies prefill only, is the exception within group (b). Position-based rules, head- or layer-level policies (FastGen, RazorAttention, DuoAttention, SimLayerKV), and the learned designs of group (c) apply one rule throughout, so they treat prompt and generated tokens alike (MoSA excepted, since its sequence-level routing has so far only been evaluated with teacher forcing). The Kept column records a second, nearly universal design choice: most methods exempt the initial (sink) tokens and/or a recent window from selection, reflecting the attention-sink phenomenon and the locality of attention, so that even query-dependent methods retain a static, query-independent core. Group (c) inherits the same prior in learned form, as the current block of MoBA and MSA, the sliding-window branch of NSA, and the local window of HiLS-Attention; MSA reports that when the forced selection of the first block and of a fixed local window is removed, the trained model still selects both, so only the block containing the query is forced in its final recipe.

Training Requirement vs. What Is Saved. The Train. and Saves columns are independent axes and should not be conflated. Training-free methods approximate the original model: their quality depends both on the backbone and on the perturbation they introduce, and they can be applied to any pretrained model at once. Trained methods change the hypothesis space and remove the train–inference discrepancy, but require training and shift the model’s representation. Between the two extremes, auxiliary-module methods (Aux) keep the LLM frozen and train only a small component, such as DuoAttention’s per-head gates or AttentionPredictor’s convolutional predictor; retrofit methods (Retro) further train an existing model to a new cache policy or attention pattern with a small fraction of the pre-training budget, as DMC and KVpop do for eviction and merging and DSA does for learned sparse attention; SepLLM, MoBA, and HiLS-Attention report both retrofit and from-scratch variants, whereas NSA, MoSA, and MSA are evaluated only as models trained from scratch. What is saved follows from how the selection mask of Eq. (17) is realized rather than from whether training was involved: removing pruned entries from the cache, or never writing them, saves memory (all of group (a), whether training-free or trained, and MoSA); skipping the unselected pairs during prefill saves attention compute, up to the cost of the selector itself (SampleAttention, SepLLM, and group (c)); restricting only what each decoding step reads saves bandwidth (group (b) and, as a by-product of a smaller cache, group (a)). Most natively sparse designs therefore require training yet save compute rather than memory, while a retrofitted eviction policy such as KVpop saves memory exactly as a training-free heuristic does.

Budget Allocation. The Budget column records the shape of the retention budget rather than its size. Most methods, from StreamingLLM to SnapKV, KVpop, and the learned designs of group (c), use one quota (U) for every layer and head. Layer-adaptive schemes (L) exploit the observation that attention concentrates on fewer tokens in deeper layers: PyramidKV fixes a decreasing schedule, DynamicKV and CAKE derive layer budgets from the attention observed at prefill, PrefixKV searches a global prefix configuration under a Lorenz-curve criterion, and SimLayerKV reduces whole

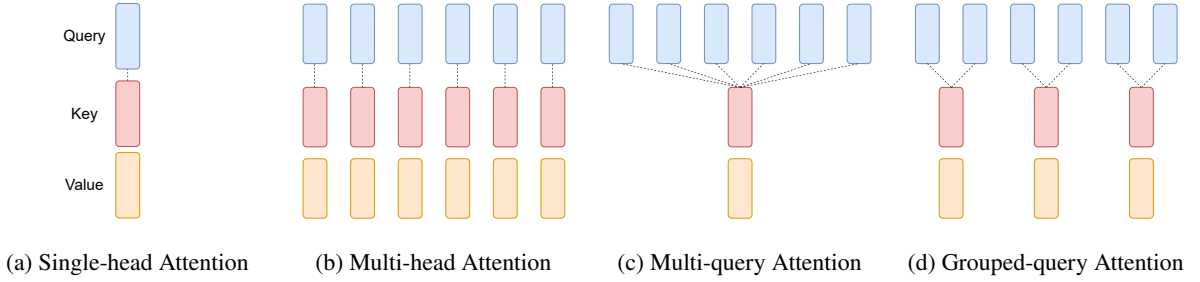


Figure 8: Optimization strategies for the number of heads in the attention mechanism.

lazy layers. Head-adaptive schemes (H) exploit the variability of attention across heads: AdaKV and CriticalKV let a layer-wide top- k set each head’s share, HeadKV, FastGen, RazorAttention, and DuoAttention fix per-head budgets or policies from a profile, LeanKV grades each head’s tokens into precision tiers, KVzip evicts the lowest reconstruction scores across heads, SampleAttention and SqueezedAttention read a head-specific number of keys, and DMC learns per-head compression ratios. Operating head by head does not by itself make an allocation head-adaptive: NACL’s random eviction and LM-Infinite’s optional per-head top- k use the same quota for every head and are therefore marked U. Most non-uniform schemes are designed for the prefill phase, where the whole prompt is visible; ReasonAlloc extends layer-wise and head-wise allocation to the decoding phase of reasoning models, where eviction methods otherwise share one budget uniformly. At an equal total budget, non-uniform allocation assigns more capacity to the layers or heads that need it, at the cost of a profiling or estimation step.

Granularity. The Gran. column records the unit over which the keep-or-read decision is made. Token-level selection (T) is the most flexible and is used by every token-eviction method of group (a) and by the token-retrieval methods of group (b); PQCache argues for it explicitly, on the grounds that relevant tokens are scattered and the contiguity assumption behind block-level lookup does not hold. Block-level selection (B) trades this flexibility for regular memory access: Quest, InfLLM, QuickLLaMA, and AttentionPredictor decide per fixed-size block or page, which maps naturally onto paged memory and fused kernels, and the natively sparse designs MoBA, NSA, MSA, and HiLS-Attention select blocks or chunks for the same reason, whereas DSA selects token-level entries and CSA selects compressed KV entries through a dedicated indexer, whose cost then motivates accelerators such as HISA and IndexCache. Cluster- and event-level units (C, E) are data-dependent: SqueezedAttention retrieves K-means clusters of keys, and EM-LLM retrieves episodic events whose variable length is set by surprise. At the coarsest end the unit is an entire head or layer (H, L): RazorAttention, DuoAttention, and SimLayerKV decide once per head or layer and add no per-step scoring cost. Whether coarser units cost accuracy relative to token-level selection at an equal budget depends on the method and task, and the reported evidence is mixed.

Summary. The three groups are complementary rather than competing, and the choice among them follows from the binding constraint. When GPU memory is the constraint and the cache cannot be offloaded, group (a) is the option for an existing model, and the remaining decisions are the phase (prompt compression for long inputs, decoding-time eviction for long outputs), the scoring signal, and the budget allocation. When quality across many queries over the same context is the constraint, group (b) keeps every option open at the cost of host memory, host-device bandwidth, and per-step estimation. When a model is being trained or continued-pretrained in any case, group (c) removes the approximation altogether and also reduces prefill and training compute, which among training-free methods only prefill-side sparsification such as SampleAttention achieves. Controlled comparisons across groups under matched budgets remain scarce, and the interaction of token reduction with the head- and dimension-oriented compression discussed next is largely unexplored.

5.2 Head Reduction

As formalized in Eq. (3), Single-head Attention (SHA) ($H = 1$, see Figure 8a) retains full head dimensionality ($d_H = d$), yet constrains representational capacity by computing a single linear combination of value matrix rows per query token. This uni-dimensional weighting limits the expressivity of the attention output $\text{Attn}(\mathbf{X})$, as all value dimensions share identical attention weights.

For the head-oriented formulas below, we write $H = H_q$ for the query-head count. Standard multi-head attention (MHA) has $H_{kv} = H$ ($H > 1$, see Figure 8b), which decomposes Query, Key, and Value matrices into H subspaces of reduced dimensionality ($d_H = d/H$). Each head independently computes attention maps of shape $[T, T]$, yielding H distinct weighting patterns over Value subvectors. Crucially, this enables dimension-specific attention distributions, enhancing contextual modeling through distinct query-dependent interactions across head subspaces.

While MHA improves representational power, caching H separate key and value heads per token per layer (a KV cache of size $2 \cdot L \cdot T \cdot Hd_H = 2LTd$ elements) imposes significant memory overhead, although at fixed total width d this is the same number of KV elements as single-head attention. Optimization strategies (Figure 8) like Multi-query Attention (MQA) [432] mitigate this by sharing Key/Value projections across heads while retaining separate Query heads (see Figure 8c). Formally, Eq. (3) can be modified as follows by removing the head index on the keys and values.

$$\mathbf{Q}^{(h)} = \mathbf{XW}_q^{(h)}, \mathbf{K} = \mathbf{XW}_k, \mathbf{V} = \mathbf{XW}_v$$

$$\text{Attn}(\mathbf{X}) = \sum_{h=1}^H \text{softmax} \left(\frac{1}{\sqrt{d_H}} \mathbf{Q}^{(h)} \mathbf{K}^\top + \mathbf{M} \right) \mathbf{VW}_o^{(h)}.$$

The primary benefit of MQA is at decoding time: the KV cache, and hence the amount of memory that must be loaded at every decoding step, shrinks by a factor of H , which directly relieves the memory-bandwidth bottleneck of incremental inference [432]; the reduction in key/value projection parameters is a secondary side effect. However, sharing a single Key and Value head reduces the flexibility of the model to learn diverse representations across heads and has been reported to degrade quality and training stability relative to MHA [1].

Grouped-query Attention (GQA) [1] is an efficient attention mechanism designed to bridge the trade-off between computational cost and model performance in Transformer architectures. It interpolates between multi-head attention (MHA) and multi-query attention (MQA) by partitioning the H query heads into g groups, where all query heads within a group share a single key and value head while retaining their own query projections (Figure 8d); $g = H$ recovers MHA and $g = 1$ recovers MQA, and the KV cache shrinks by a factor of H/g relative to MHA. Here $H_{kv} = g$, so the cache has $2LTgd_H$ elements; at fixed total width $d = Hd_H$, standard SHA and MHA both cache $2LTd$ elements.

$$\mathbf{Q}^{(h)} = \mathbf{XW}_q^{(h)}, \mathbf{K}^{(\lceil hg/H \rceil)} = \mathbf{XW}_k^{(\lceil hg/H \rceil)}, \mathbf{V}^{(\lceil hg/H \rceil)} = \mathbf{XW}_v^{(\lceil hg/H \rceil)}$$

$$\text{Attn}(\mathbf{X}) = \sum_{h=1}^H \text{softmax} \left(\frac{1}{\sqrt{d_H}} \mathbf{Q}^{(h)} \mathbf{K}^{(\lceil hg/H \rceil)\top} + \mathbf{M} \right) \mathbf{V}^{(\lceil hg/H \rceil)} \mathbf{W}_o^{(h)}.$$

Here $\lceil \cdot \rceil$ is the ceiling function and g is the number of groups. When $1 < g < H$, GQA reduces memory usage compared to full MHA, while compared to MQA, it maintains some level of diversity by allowing different groups to have different keys and values. GQA aims to strike a balance between efficiency and model expressiveness, making it useful for applications where resource constraints are a concern but some level of flexibility is still desired.

In addition to MQA and GQA, AsymGQA [50] proposes an optimized, activation-informed asymmetric grouping strategy for converting pretrained MHA models into hardware-efficient GQA models. Unlike GQA using uniform grouping and neighbor-based head merging, AsymGQA employs a similarity-guided stochastic search algorithm to group KVs asymmetrically (allowing variable group sizes) based on activation patterns (measured via cosine similarity of layer outputs), not just weight proximity. The grouping is informed by similarity in attention-head activations. In addition, WGQA [57] enhances standard GQA by introducing learnable weights for KV heads during fine-tuning. These weights, initialized as group-averages or randomly, enable a parametric weighted aggregation of KV heads in decoder attention blocks. WGQA exhibits near MHA performance with no inference overhead as weights are absorbed post-fine-tuning. Furthermore, QCQA [213] proposes an evolutionary algorithm to identify optimal query head groupings. Unlike GQA's equal-sized groups, QCQA enables arbitrary-sized groupings or constrained equal-sized groups across layers. It uses a computationally cheap fitness function, Weight-Sharing Error (WSE), measuring mean squared error between original and merged KV weight distributions, as a proxy for accuracy loss, avoiding costly LLM evaluations. Similarly, KDGQA [222] introduces a dynamic extension of GQA that improves query-to-group allocation by leveraging the L_2 -norms of key vectors as a proxy for their importance. Unlike GQA, KDGQA allocates queries to groups based on the relative magnitudes of the key vectors within each layer. The allocation is performed during each forward pass, ensuring that the grouping mechanism adapts to the importance of key vectors.

Head reduction methods primarily adjust how query heads share a smaller set of KV heads, effectively optimizing the query-to-KV head ratio (e.g., MHA $\rightarrow$ GQA/MQA) to reduce KV cache storage while preserving accuracy. The total number of heads is usually predetermined manually, which does not increase with the length of the input text, or the complexity of the task. Therefore, head methods are not as numerous or diverse as those targeting the number of tokens. Currently, the most commonly used method in engineering is GQA due to its ease of implementation and relatively robust performance.

5.3 Dimension Reduction

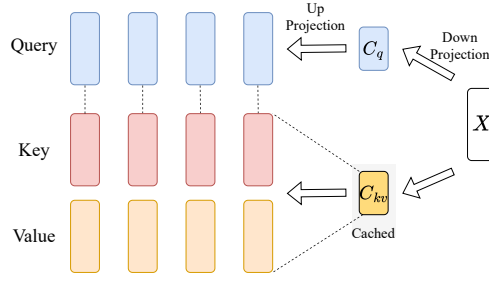


Figure 9: Illustration of dimension reduction: MLA caches a shared low-dimensional latent per token and recovers per-head keys and values by up-projection.

Beyond conventional reduction strategies targeting token and head, a third axis compresses the *width* of the cached per-token representation. Multi-head Latent Attention (MLA) [73] is the architecture-level instance of this idea. The basic idea is a low-rank joint compression of keys and values: each token’s keys and values for all heads are generated from a single low-dimensional latent vector via a down-projection, and this latent is cached along with a small positional component when decoupled RoPE is used; the per-head keys and values are recovered by up-projection when needed, as shown in Figure 9. Note that MLA does not shrink the per-head dimension d_H used inside attention; it replaces H separate key and value heads with one shared latent whose dimension d_c is much smaller than $2Hd_H$. At inference time the up-projection matrices can be absorbed into the query and output projections, so that attention scores are computed directly against the cached latents without materializing per-head keys and values. Through this strategy, MLA substantially reduces the size of the KV cache and the memory traffic of decoding while maintaining quality comparable to MHA in the reported experiments [73].

For the following schematic derivation, we use a common latent width d_c for keys/values and queries for notational simplicity; actual models may choose different widths. MLA has the following expression for keys and values:

$$\mathbf{C}_{kv} = \mathbf{X}\mathbf{W}_{dkv},$$

$$\mathbf{K}^{(h)} = \mathbf{C}_{kv}\mathbf{W}_{uk}^{(h)}, \mathbf{V}^{(h)} = \mathbf{C}_{kv}\mathbf{W}_{uv}^{(h)},$$

where $\mathbf{W}_{dkv} \in \mathbb{R}^{d \times d_c}$ is the down-projection matrix for keys and values, d_c is the compression dimension ($d_c \ll 2Hd_H$) and $\mathbf{W}_{uk}^{(h)}, \mathbf{W}_{uv}^{(h)} \in \mathbb{R}^{d_c \times d_H}$ are the up-projection matrices for the h th head. Furthermore, MLA also applies low-rank compression for queries in order to reduce the activation memory during training:

$$\mathbf{C}_q = \mathbf{X}\mathbf{W}_{dq}, \mathbf{Q}^{(h)} = \mathbf{C}_q\mathbf{W}_{uq}^{(h)},$$

where $\mathbf{W}_{dq} \in \mathbb{R}^{d \times d_c}$ and $\mathbf{W}_{uq}^{(h)} \in \mathbb{R}^{d_c \times d_H}$ are down-projection and up-projection for queries respectively.

It is worth noting that MLA cannot be seamlessly combined with RoPE [450] in its naive form. The incompatibility concerns the weight-absorption trick described above: RoPE inserts a position-dependent rotation between the query projection and the key up-projection $\mathbf{W}_{uk}$, and because this rotation depends on the relative position of the query and key, it cannot be folded into a single static matrix. Without absorption, the keys of all cached positions would have to be up-projected and rotated at every decoding step, forfeiting the computational advantage of attending directly over the latents (the storage saving itself would remain). Consequently, MLA adopts a decoupled approach: a small additional rotary key, shared across heads, and a matching rotary query component carry the positional information, while the main low-rank latent remains position-free and absorbable [73].

Subsequent work on this axis splits into native alternatives to MLA, conversions of existing models, and post-training compression of the hidden dimension. Tensor Product Attention (TPA) [567] factorizes the query, key, and value *activations* of each token, rather than a static weight, as a sum of contextual tensor products: $\mathbf{Q}_t = R_Q^{-1} \mathbf{A}_Q(\mathbf{x}_t)^\top \mathbf{B}_Q(\mathbf{x}_t)$ and likewise for $\mathbf{K}_t$ and $\mathbf{V}_t$, where the factor matrices are linear functions of the hidden state. Only the key and value factors are cached, so the per-token footprint is set by the ranks R_K and R_V rather than by Hd_H . RoPE is applied to the $\mathbf{B}_Q$ and $\mathbf{B}_K$ factors and is therefore native; there is no need for MLA’s decoupled rotary stream. The authors report that a TPA Transformer (T6) matches or improves upon MHA, MQA, GQA, and MLA in pre-training perplexity and downstream scores at a substantially smaller KV cache.

Grouped Latent Attention (GLA) [548] keeps MLA’s latent compression but changes how the latent is laid out for hardware. MLA’s single latent cannot be sharded, so every device in tensor parallelism must reload the full cache. GLA groups the latent into shardable slices of width $2d_H$ and pairs this with an optimized kernel; it is reported to match MLA quality while being easier to shard, with up to $2\times$ the throughput of FlashMLA when the query length exceeds one (as in speculative decoding) and up to $2\times$ end-to-end serving throughput from the smaller per-device load. The same paper introduces Grouped-Tied Attention (GTA), which ties the key and value of each GQA group into one state and applies RoPE only to a small extra key; GTA is reported to match GQA quality at roughly half the cache.

MLA and TPA must be trained (or continued-pretrained) as part of the model. TransMLA [293] instead converts a pretrained GQA or MHA model into an MLA model by initializing the latent projections from the existing weights, decoupling RoPE so that absorption remains valid, and fine-tuning. On LLaMA-2-7B it reports compressing 93% of the KV cache and a $10.6\times$ inference speedup at an 8K context after fine-tuning on 6B tokens, and the converted checkpoint is compatible with DeepSeek-style MLA serving stacks.

A third line never changes the attention definition and only compresses the hidden dimension of an already-trained cache. Palu [43] applies SVD offline to the key and value projection weights, caches the low-rank intermediate, and reconstructs full keys and values at decode time. A group-head decomposition (G-LRD) interpolates between per-head SVD, which is cheap to reconstruct but less accurate, and a single joint SVD across all heads, which is more accurate but H times more expensive to reconstruct; ranks are assigned from Fisher information. Palu reports a 50% KV cache reduction on popular open models and is orthogonal to quantization and token eviction. Because RoPE sits between the reconstructed key and the query, Palu cannot absorb the reconstruction into a static matrix the way MLA does, so the storage saving does not automatically become a latency saving unless the reconstruction, RoPE, and score are fused.

ThinK [529] reduces width by pruning channels of the *key* cache rather than by a low-rank subspace. It scores channel j of head i by $\|\mathbf{Q}_{:,j}^{(i)} (\mathbf{K}_{:,j}^{(i)})^\top\|_F$ over a query window and greedily keeps the top channels, leaving values intact. The method is training-free and complementary to token eviction and KV quantization; combined with them it is reported to add more than 20% extra memory reduction on LLaMA and Mistral long-context benchmarks. Unlike MLA, TPA, and Palu, ThinK does not introduce a latent: it simply drops coordinates of $\mathbf{K}$.

5.4 Layer Reduction

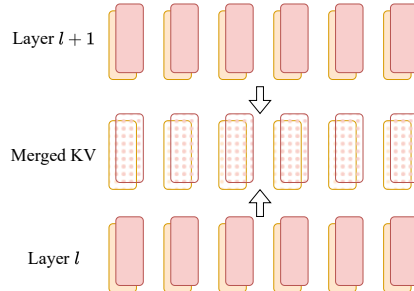


Figure 10: Illustration of cross-layer KV sharing: adjacent layers reuse or merge their KV states.

The last perspective of reduction is layer reduction, which shares or merges KV states across layers (Figure 10). MiniCache [270] proposes a method to compress the KV pairs of two adjacent Transformer layers together, based on the observation that the KV cache states between two adjacent layers are highly similar, particularly across the middle to

deep layers. Specifically, it merges the KV caches of two subsequent layers into shared states and selects unmergeable tokens for retention. In addition to MiniCache, several concurrent works share KV heads across layers rather than merging them post hoc. Cross-Layer Attention (CLA) [32] lets a group of adjacent layers reuse the keys and values computed by one of them, and YOCO [457] caches keys and values only once in a self-decoder and lets a cross-decoder attend to this shared cache. MLKV [591] combines this idea with MQA: it uses a single KV head within each layer and further shares that head across multiple self-attention layers. Moreover, LCKV [512] proposes computing and caching KV pairs for only a small subset of layers, often just the upper layer. During inference, lower-layer queries reuse these cached KVs, reducing KV cache memory and improving decoding throughput. While LCKV [512] introduces dependencies on top-layer cached values from previous tokens (conflicting with the parallel training of Transformers), it addresses this challenge with an approximate training method that preserves parallelism, making LCKV a practical solution for reducing resource overhead while maintaining competitive performance. In contrast to focusing on sharing KV across layers, Shared Attention [261] targets sharing the attention scores directly. SA proposes to reuse precomputed attention weights across multiple layers, rather than recalculating them for each layer. This method takes advantage of the isotropic patterns often observed in the attention distributions of pre-trained LLMs, where attention structures remain consistent across layers. By directly sharing these attention weights, SA avoids recomputing attention scores in the sharing layers, reducing both computation and the amount of key cache those layers require.

5.5 Future Directions

In this section, we have discussed efficient attention from four perspectives: token, head, dimension, and layer. Each of these approaches can be viewed as a form of compression, aimed at improving the efficiency of training or inference, and is motivated by the observation that KV caches exhibit redundancy along multiple axes. Two directions appear particularly promising:

- **Input-Dependent Token Methods:** Among these axes, token reduction stands out as the most complex and context-sensitive, since input tokens are inherently dynamic and variable in length. Many head-, dimension-, and layer-sharing designs use fixed architectural choices, although input-dependent allocations such as KDQA provide exceptions. Token methods therefore offer one important route to input-dependent state and access control.
- **Composing Multiple Axes:** A promising direction is to integrate several of these axes within a single architecture. The axes are not fully independent, however: MLA already shares one latent across all heads, so it cannot be stacked with GQA-style head grouping or with head-wise budget allocation (e.g., AdaKV) in a straightforward way, and cross-layer sharing constrains which layers can be pruned independently. Understanding which combinations compose cleanly, and at what cost in quality, remains an open question.

6 Linear Language Models

Compared with efficient attention, linear recurrent/attention models have emerged as another category of prominent alternatives. These models are built on operators with linear computational complexity, making them more efficient at processing long sequences. This section examines the mathematical formulations of their basic operators, grouped into four types: recurrent neural networks, fast weight programmers, state space models, and linearized attention. For each category, we trace their historical development and elaborate on their specific formulations. The per-timestep computation graphs of five representative operators are illustrated in Figures 11a to 11e.

6.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have a long and influential history in machine learning [414, 417, 405]. Unlike conventional feed-forward neural networks, RNNs introduce *feedback connections* that form directed cycles within the network. These pathways allow information from previous timesteps to influence future computations, endowing the model with an internal state capable of retaining contextual information. Such temporal memory is crucial for tasks involving sequential or time-dependent data, including speech and language modeling [323, 94, 421, 388, 100, 27, 116, 301, 302, 128, 460].

The conceptual roots of recurrent computation can be traced to early dynamical systems such as the *Ising model* (or Lenz-Ising model) [245, 197, 237, 506, 232] in the 1920s, which illustrated how interacting units with cyclic

dependencies can maintain and stabilize internal states [405]. Later, foundational works [289, 232] proved that such recurrent structures could function as formal computational units capable of processing sequences. Although such systems lacked trainable mechanisms, they offered an early template for later recurrent neural models that rely on feedback connections to maintain temporal context. A standard recurrent neural network formalizes this notion of temporal memory by updating a hidden state based on both the current input and the previous hidden state. Mathematically, the basic operation of a vanilla RNN is expressed as:

$$\begin{aligned} \mathbf{h}_t &= f(\mathbf{W}, \mathbf{h}_{t-1}, \mathbf{x}_t), \\ \mathbf{y}_t &= \mathbf{h}_t \mathbf{W}_y, \end{aligned} \quad (19)$$

where $\mathbf{x}_t$ is the input at timestep t , $\mathbf{h}_t$ is the recurrent hidden state (memory), $\mathbf{y}_t$ is the output, and $f(\cdot)$ is a nonlinear transition function parameterized by $\mathbf{W}$. The hidden state integrates past information through $\mathbf{h}_{t-1}$, and the output is produced by decoding $\mathbf{h}_t$ with weights $\mathbf{W}_y$. In autoregressive settings, $\mathbf{y}_t$ may be fed back as part of the next input, enabling sequence generation. Since the hidden state has fixed dimensionality, an RNN retrieves historical information solely through this compressed memory rather than storing all past inputs explicitly. This makes each recurrent update constant-time, yielding overall linear complexity in sequence length. Such computational efficiency has historically positioned RNNs as a central approach for modeling long-context sequential data.

Trainable RNNs first emerged in the early 1970s. Pioneering work by Shun-Ichi Amari in 1972 made the Ising recurrent architecture adaptive, introducing learning rules that modified recurrent connections to associate input patterns with output patterns by dynamically adjusting connection weights [6, 170, 324]. Around the same time, recurrent computation in the context of biological neural dynamics and hippocampal memory was explored [133, 287]. Subsequent research on associative memory and self-organizing recurrent structures further advanced this direction [134, 235, 324]. Then, *Backpropagation Through Time* (BPTT) [268, 394, 507] was formalized as a general method for training RNNs, enabling gradient-based optimization of model parameters. However, this training procedure revealed two fundamental challenges: the *vanishing and exploding gradient problems* [165], wherein gradients either decay toward zero or grow exponentially as they are propagated backward through long temporal sequences.

To address the vanishing and exploding gradient problem, the concept of *recurrent residual connection* was proposed [165], where a neural unit with the identity activation function has a connection to itself and the weight of this connection is 1.0 [418]. This principle was instantiated in the standard Long Short-Term Memory (LSTM) architecture in 1997 [167]. The core LSTM unit featuring this residual connection was termed *constant error carousels* (CECs) [167], designed to enforce the constant error flow. The specific design is as follows:

$$\begin{aligned} \mathbf{c}_t &= \mathbf{c}_{t-1} + \mathbf{g}_t \odot \mathbf{m}_t, \\ \mathbf{h}_t &= \mathbf{n}_t \odot \tanh(\mathbf{c}_t), \end{aligned} \quad (20)$$

where $\mathbf{c}_t$ is the cell state (memory) on which operations are performed, $\mathbf{m}_t$ represents the processed new information to be added, $\mathbf{g}_t$ denotes the input gate activation controlling how much new information is added to the cell state, $\mathbf{n}_t$ denotes the output gate activation controlling how much of the cell state is exposed to the next step, and $\odot$ denotes element-wise multiplication. Subsequently, the later LSTM with a forget gate [117] introduced the adaptive *forget gate activation* $\mathbf{f}_t$ to replace the fixed weight, forming the *gated recurrent residual connection*. Thus, the update rule becomes:

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{g}_t \odot \mathbf{m}_t, \\ \mathbf{h}_t &= \mathbf{n}_t \odot \tanh(\mathbf{c}_t). \end{aligned} \quad (21)$$

Although the learnable forget gate may reintroduce the vanishing gradient problem, it is rendered more controllable via training: when $\mathbf{f}_t$ approaches 1, it approximates constant error flow; when necessary (e.g., to shift focus to new tasks), it can temporarily disrupt this flow. By incorporating gating mechanisms and a dedicated cell state, LSTMs enable selective retention or forgetting of past information during the forward pass. The gate activations as well as the input transformation are computed as:

$$\begin{aligned} \mathbf{f}_t &= \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_f), \\ \mathbf{g}_t &= \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_g), \\ \mathbf{m}_t &= \tanh([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_m), \\ \mathbf{n}_t &= \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \mathbf{W}_n), \end{aligned} \quad (22)$$

where $[\cdot, \cdot]$ denotes concatenation, σ is the sigmoid activation function, and $\mathbf{W}_*$ are the corresponding weight matrices.

To further stabilize the gradients in RNNs, various initialization techniques for their weight matrices have proven effective. For instance, initializing these matrices as identity matrices [243] or orthogonal matrices [400] can lead to a positive impact. A classic example of this principle is the later LSTM with a forget gate [117], where forget gates were specifically initialized to 1.0. This ensures the gates remain open at the start of training, effectively allowing the model to begin with plain residual connection, like the 1997 LSTM. Additionally, injecting special structural biases can also be beneficial, especially for long sequences, including: (a) unitary and orthogonal model weight matrices [155, 11]; (b) exponential parameterization for diagonal weight matrices [139, 343]; (c) time (positional) decay mechanisms [352].

In addition to stabilizing backpropagation, enabling parallel training is also crucial for the practical deployment of RNNs, particularly in large language models. However, the inherent sequential nature of RNN hidden states prevents parallel computation. A key strategy to overcome this limitation is to *linearize the temporal dependencies* of the inputs [288], leading to the development of *linear RNNs*. Recently, significant efforts have focused on scaling the language models formed by linear RNNs for large-scale training:

- The Linear Recurrent Unit (LRU) [343] improves the training of large linear RNNs by employing exponential parameterization of complex diagonal weight matrices, γ -normalization of hidden activations, and weight initialization with restricted eigenvalue phases. Hawk [72] further shows that incorporating gating mechanisms into LRUs can lead to performance improvements.
- Receptance Weighted Key Value (RWKV) [352] is another prominent large-scale RNN architecture. Each RWKV block comprises a token mixing operation and a channel mixing operation, both of which perform local feature aggregation and transformation on sequential inputs, followed by a gated linear unit. Note that the token mixing operation incorporates a linear recurrent scan of the transformed features before the gating unit to enable global information aggregation, alongside time (positional) decay and specialized weight initialization for stabilizing training. While this architectural shift to linear recurrence enables parallel training, it introduces expressivity trade-offs. RWKV as well as Mamba [136] remains a simplified and less powerful variant of the LSTM [117], streamlining the gating mechanisms for efficiency at the expense of the fine-grained state-tracking inherent in the original LSTM.
- xLSTM [24] revisits the LSTM architecture for large language models, using sLSTM and mLSTM modules as its backbone—both enhanced variants of LSTM. These modules employ exponential gating to improve storage decision revisions (i.e., finding the most relevant past tokens for a given query). The sLSTM extends the scalar memory of the vanilla LSTM by introducing a memory mixing module, which establishes a connection between the two non-linear recurrent components—the cell memory and the hidden state. To enhance capacity, sLSTM extends this mechanism to multiple heads, where mixing occurs independently within each head but not across them. However, due to the non-linear nature of its recurrence, sLSTM does not support parallelization. In contrast, mLSTM abandons memory mixing to enable parallelization, effectively adopting the linear recurrent principles originally established by ULTRA [406, 408]. It adopts a matrix-based memory for enhanced capacity. This memory is updated using a covariance (outer product) update rule, with multiple heads and cells treated equivalently in the multi-head scenarios.

6.2 Fast Weight Programmers

Fast Weight Programmers (FWPs) [406, 408, 412, 401, 402, 403, 190, 192, 185] are architectures where a slow neural network dynamically programs the fast weights [490, 97, 163] of other networks. Typically, an FWP consists of two interacting components: (1) a slow network that receives the current input and produces programming signals, and (2) a fast network whose parameters are updated at every time step according to these signals. This design yields an explicit separation between control and storage, where the control is implemented by the slow weights that learn slowly across tasks while the storage is implemented by fast weights that store short-lived information within the input sequence, serving as an efficient memory. FWPs can also be used for meta-learning with self-referential NNs that can learn to execute and modify their own weight change algorithm [407, 413, 410, 411, 228, 404, 193, 229, 195].

A specific instance of FWP with a differentiable update rule is formulated as follows [406, 194]:

$$\begin{aligned}
 \mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t &= \mathbf{x}_t \mathbf{W}^{\text{slow}}, \\
 \mathbf{W}_t &= f(\mathbf{W}_{t-1}, \mathbf{k}_t \otimes \mathbf{v}_t), \\
 \mathbf{y}_t &= \mathbf{q}_t \mathbf{W}_t,
 \end{aligned} \tag{23}$$

where $\mathbf{W}^{\text{slow}}$ and $\mathbf{W}_t$ denote the slow weights and fast weights, respectively, the index t denotes the timestep of a series of data, f denotes a differentiable (e.g., sum-and-squash) function, and $\otimes$ denotes the outer product. The slow network first projects the current input into query, key, and value vectors. The outer product of the key and value vectors is then used to update the fast weights sequentially. At each timestep, the updated fast weights are queried using the current query to produce the output.

The above form of FWP with a sum aggregation function is equivalent to the linear Transformer without normalization [403] (detailed in Sec. 6.5), a connection that has led to it being retrospectively identified as the Unnormalized Linear Transformer (ULTRA) [406, 408, 420, 403]. This perspective has reignited interest in FWPs within the language modeling community. Notably, DeltaNet [403] replaces the additive outer-product update rule with an error-correcting delta rule [508]. This enables it to more readily learn corrections to the current key–value mappings, thereby improving the memorization of key–value associations. The specific formulation of the delta rule is given as follows:

$$\begin{aligned}\bar{\mathbf{v}}_t &= \phi(\mathbf{k}_t) \mathbf{W}_{t-1} \\ \mathbf{v}_t^{\text{new}} &= \bar{\mathbf{v}}_t + \beta_t (\mathbf{v}_t - \bar{\mathbf{v}}_t), \\ \mathbf{W}_t &= \mathbf{W}_{t-1} + \phi(\mathbf{k}_t) \otimes \mathbf{v}_t^{\text{new}} - \phi(\mathbf{k}_t) \otimes \bar{\mathbf{v}}_t \\ &= \mathbf{W}_{t-1} + \beta_t \phi(\mathbf{k}_t) \otimes (\mathbf{v}_t - \bar{\mathbf{v}}_t),\end{aligned}\tag{24}$$

where $\phi(\cdot)$ is the kernel function discussed in Sec. 6.4, $\bar{\mathbf{v}}_t$ is the value retrieved from the previous fast weights at the current key, and β_t controls how far it is moved toward $\mathbf{v}_t$. Thus, $-\phi(\mathbf{k}_t) \otimes \bar{\mathbf{v}}_t$ is the removal term constructed from the current key and its retrieved value, whereas $+\phi(\mathbf{k}_t) \otimes \mathbf{v}_t^{\text{new}}$ is the write term pairing the same key with the interpolated replacement. We refer below to these key-selective operations as erasing and writing, reserving forgetting for a decay applied to the entire previous state. The effectiveness of the delta rule as a state update rule has been demonstrated across many applications, including language modeling [403, 190, 540, 541], reinforcement learning in game environments [190, 193], time series classification [192], and image generation [185]. Remarkably, the delta rule also improves the expressivity of FWPs compared to FWPs with purely additive or decay-based update rules [194, 131]. DeltaProduct [438] extends DeltaNet by applying multiple delta-rule steps per token, yielding state-transition matrices formed as products of generalized Householder transformations and improving state tracking and length extrapolation.

Such a rule can be derived [508] as the gradients of the mean squared error between $\mathbf{v}_t$ and $\phi(\mathbf{k}_t) \mathbf{W}_t$ —akin to the core of the revived test-time training models [454]—making it highly effective for in-context retrieval tasks. Gated DeltaNet [541] combines DeltaNet’s key-selective replacement with a data-dependent scalar decay of the entire previous state, complementing targeted key–value updates with a mechanism for rapidly clearing outdated information. Comba [172] controls global forgetting and current-key erasure separately: a learned ratio in $(0, 1)$ keeps the erasure coefficient below the write coefficient, and the current key is also used to adjust the readout. RWKV-7 [354] introduces a generalized delta rule with vector-valued gating and in-context learning rates. These mechanisms allow different state features to decay and be corrected at different input-dependent rates. Its relaxed value replacement rule further allows the erase and write terms to use different directions and strengths. Kimi Delta Attention (KDA) [471] replaces Gated DeltaNet’s scalar decay with channel-wise decay, allowing different memory subspaces to operate at different retention timescales, and serves as the recurrent linear-attention operator in Kimi Linear. Since KDA still uses a shared scalar step size for erasing and writing, Gated DeltaNet-2 [149] replaces it with separate channel-wise key-side erase and value-side write gates.

6.3 State Space Models

A continuous-time linear State Space Model (SSM) [436] expresses both the rate of change of the state and the output as linear functions of the current state and input [135, 139]:

$$\begin{aligned}\frac{d}{dt} \mathbf{h}(t) &= \mathbf{h}(t) \mathbf{A} + \mathbf{x}(t) \mathbf{B}, \\ \mathbf{y}(t) &= \mathbf{h}(t) \mathbf{C} + \mathbf{x}(t) \mathbf{D},\end{aligned}\tag{25}$$

where $\mathbf{x}(t)$, $\mathbf{h}(t)$, and $\mathbf{y}(t)$ denote the input, state, and output at time t , respectively, and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are the model weight matrices. The choice of $\mathbf{A}$ and $\mathbf{B}$ in Eq. (25) determines how $\mathbf{h}(t)$ represents the input history. For instance, HiPPO [137] approximates this history using a fixed number of polynomial projection coefficients stored in $\mathbf{h}(t)$. Differentiating these coefficients yields a linear state equation with potentially time-dependent matrices $\mathbf{A}$ and $\mathbf{B}$.

The SSM also admits a discrete-time recurrent form:

$$\begin{aligned} \mathbf{h}_t &= \mathbf{h}_{t-1} \bar{\mathbf{A}} + \mathbf{x}_t \bar{\mathbf{B}}, \\ \mathbf{y}_t &= \mathbf{h}_t \mathbf{C} + \mathbf{x}_t \mathbf{D}. \end{aligned} \quad (26)$$

The discrete form in Eq. (26) can be derived from the continuous form in Eq. (25) through discretization with step size $\Delta > 0$. For example, zero-order hold (ZOH) gives $\bar{\mathbf{A}} = \exp(\Delta \mathbf{A})$ and $\bar{\mathbf{B}} = \Delta \mathbf{B}(\Delta \mathbf{A})^{-1} (\exp(\Delta \mathbf{A}) - \mathbf{I})$ for invertible $\mathbf{A}$. The state update in Eq. (26) corresponds to an affine choice of f in the memory update of Eq. (23), and thus the discretized linear SSM is equivalent to a linear RNN (Sec. 6.1). This equivalence also carries over to model design, i.e., the techniques used in discrete SSMs can also be applied to the corresponding recurrent models, including parallelization computation [139], diagonal state parameterizations and initialization [141, 138], multi-input multi-output states and multidimensional extensions [439, 327], or input-dependent transitions and gated activations [148, 290].

Mamba [136] is a competitive discrete SSM (linear RNN) backbone for language modeling, extending the time-invariant recurrence in Eq. (26) with content-dependent memory control to better capture long-range dependencies. It makes $\bar{\mathbf{A}}$, $\bar{\mathbf{B}}$, and $\mathbf{C}$ input-dependent by generating $\mathbf{B}_t$, $\mathbf{C}_t$, and the discretization step Δ_t from the input at timestep t ; the resulting operators are denoted by $\bar{\mathbf{A}}_t$, $\bar{\mathbf{B}}_t$, and $\mathbf{C}_t$. This selection mechanism draws on LSTM’s gating principle [167, 117]: $\bar{\mathbf{A}}_t$ determines what is retained from the previous state, $\bar{\mathbf{B}}_t$ controls what is written from the current input, and $\mathbf{C}_t$ controls what is read out, while $\mathbf{D}$ provides a learned, input-independent skip connection from input to output.

Mamba-2 [70] improves Mamba’s parallel training efficiency on long sequences with tensor-core-accelerated matrix multiplications. To enable this acceleration, it constrains $\bar{\mathbf{A}}_t$ to a scalar multiple of the identity, so its core recurrence can be written as additive ULTRA with scalar forgetting (Sec. 6.2) [541], or equivalently as a scalar-gate specialization of Gated Linear Attention (GLA) [539]. For parallel training, Mamba-2 computes the unrolled recurrence (a sum of writes weighted by transition products) using intra-chunk parallelism and inter-chunk recurrence. Within each chunk, it combines $\mathbf{C}_t$ and $\mathbf{B}_t$ for $i \leq t$, incorporates decay weights, and applies the resulting chunk-local interaction matrix to the scaled inputs. Together with inter-chunk recurrence, this avoids explicitly forming and multiplying by the full sequence interaction matrix. Between chunks, the state follows the affine update structure of Eq. (26). These recurrent and parallel computations are linked by the semiseparable structure formalized in Structured State-Space Duality (SSD) [70].⁴ Afterwards, Mamba-3 [240] continues this line with an improved SSM discretization, complex-valued state transitions for richer state tracking, and a multi-input multi-output update designed to improve the quality–efficiency trade-off.

The real-valued Mamba has been shown to fail on simple state-tracking tasks such as parity [398, 131]. This limitation arises from the nonnegative diagonal structure of $\bar{\mathbf{A}}_t$, which prevents parity recognition at arbitrary sequence lengths in the fixed-depth, finite-precision setting. To address this limitation, Grazi et al. [131] extend the transition range to include negative values, enabling the modified Mamba to learn parity. Its real diagonal structure, however, still prevents it from counting modulo three under the same assumptions. In contrast, nonlinear RNNs such as LSTM and sLSTM can in principle track any finite-state machine, but their standard recurrences require sequential computation [167, 24, 131]. An alternative is to improve state tracking through more expressive linear transitions while retaining parallel training. For example, allowing negative eigenvalues in DeltaNet enables reflection-like state transitions and improves its ability to track sequences of permutations [131]. RWKV-7 [354] further generalizes the delta rule with channel-wise control and more flexible erase–write operations. Its theoretical analysis provides a four-layer construction that recognizes any regular language under the stated parameterization and log-precision assumptions. These delta-rule models retain updates that are affine in the previous state, allowing stronger state tracking without sacrificing parallel training.

6.4 Linearized Attention

Linearized attention denotes a category of methods improving the computational efficiency of the dot-product attention mechanism in the Transformer [489]. Its formulation can be derived by approximating the softmax function with linear operations. These methods lower the complexity from quadratic to linear when calculating pairwise token similarity scores. Historically, the conceptual foundation of this approach can be traced back to the Unnormalized Linear Transformer (ULTRA) [406, 408, 420, 419]. Linearized attentions are closely connected to linear recurrent operators, paving the way for further developments in linear recurrent models.

Derivation of Linearized Attention. Intuitively, linear complexity can be achieved by reordering the computation of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ in Eq. (3). Nonetheless, this rearrangement is non-trivial because the softmax function has no multiplicative

⁴Related classical primal–dual views of linear models are discussed in Aizerman [3], Irie et al. [191].

Method	Feature map $\phi(\mathbf{x})$
Linear Attention [219]	$\text{elu}(\mathbf{x}) + \mathbf{1}$
Cosformer [363]	$\text{ReLU}(\mathbf{x})$
Performer’s Kernel [58]	$\frac{g(\mathbf{x})}{\sqrt{D}} [f_1(\mathbf{w}_1 \mathbf{x}^\top), \dots, f_1(\mathbf{w}_D \mathbf{x}^\top), \dots, f_l(\mathbf{w}_1 \mathbf{x}^\top), \dots, f_l(\mathbf{w}_D \mathbf{x}^\top)]$
Random Feature Attention [356]	$\frac{1}{\sqrt{D}} [\sin(\mathbf{w}_1 \mathbf{x}^\top), \dots, \sin(\mathbf{w}_D \mathbf{x}^\top), \cos(\mathbf{w}_1 \mathbf{x}^\top), \dots, \cos(\mathbf{w}_D \mathbf{x}^\top)], \mathbf{w}_i \sim \mathcal{N}(0, \beta \mathbf{I})$
FAVOR+ [58]	$\frac{\exp(-\frac{\ \mathbf{x}\ _2^2}{2})}{\sqrt{2D}} [\exp(\mathbf{r}_1 \mathbf{x}^\top), \dots, \exp(\mathbf{r}_D \mathbf{x}^\top), \exp(-\mathbf{r}_1 \mathbf{x}^\top), \dots, \exp(-\mathbf{r}_D \mathbf{x}^\top)], \mathbf{r}_i \sim \mathcal{N}(0, \mathbf{I})$
Hedgehog [559]	$[\exp(\mathbf{w}_1 \mathbf{x}^\top + \mathbf{b}_1), \dots, \exp(\mathbf{w}_D \mathbf{x}^\top + \mathbf{b}_D)]$
Based [12]	$[1, x_1, \dots, x_{d_H}, x_1^2/\sqrt{2}, \dots, x_{d_H}^2/\sqrt{2}, x_1 x_2, x_1 x_3, \dots, x_{d_H-1} x_{d_H}]$
ReBased [4]	$(\mathbf{a} \odot \text{LayerNorm}(\mathbf{x}) + \mathbf{b})^2$

Table 3: The representative choices of the feature map ϕ in linearized attention; the induced kernel is $\phi(\mathbf{q})\phi(\mathbf{k})^\top$. The notations f and g denote arbitrary functions, while β represents a scalar hyperparameter. D is the feature dimension of ϕ (the number of random features), l is the number of feature functions in Performer’s feature map, and x_j denotes the j -th coordinate of $\mathbf{x} \in \mathbb{R}^{d_H}$. ReBased learns the affine parameters $\mathbf{a}, \mathbf{b}$ of the normalized features separately for queries and keys.

property. To address this, one straightforward approach is to substitute the softmax with other decomposable functions. The earlier works [488, 219] view the unnormalized part of the softmax, $\exp(\mathbf{q}_t \mathbf{k}_i^\top)$, as a positive-definite kernel evaluated on the query and the key, where $\mathbf{q}_t$ and $\mathbf{k}_i$ denote the t -th and i -th row vectors of the matrices $\mathbf{Q}$ and $\mathbf{K}$, respectively. Every such kernel is the inner product of a feature map, $\exp(\mathbf{q}_t \mathbf{k}_i^\top) = \phi(\mathbf{q}_t)\phi(\mathbf{k}_i)^\top$, but for the exponential kernel the exact feature map is infinite-dimensional, so no finite-dimensional ϕ satisfies this equality exactly [219]. Linearized attention therefore works with a finite-dimensional feature map $\phi : \mathbb{R}^{d_H} \rightarrow \mathbb{R}^D$, where D is the feature dimension, chosen either so that $\phi(\mathbf{q}_t)\phi(\mathbf{k}_i)^\top \approx \exp(\mathbf{q}_t \mathbf{k}_i^\top)$ (random-feature or Taylor approximations) or as a different decomposable kernel that replaces the exponential altogether. With such a ϕ , each output vector of the linearized attention module is formulated as follows:

$$\begin{aligned}
\mathbf{y}_t = \text{CausalLinearAttn}(\mathbf{q}_t, \mathbf{K}, \mathbf{V}) &= \frac{\sum_{i=1}^t \phi(\mathbf{q}_t)\phi(\mathbf{k}_i)^\top \mathbf{v}_i}{\sum_{i=1}^t \phi(\mathbf{q}_t)\phi(\mathbf{k}_i)^\top} \\
&= \frac{\phi(\mathbf{q}_t) \cdot \sum_{i=1}^t \phi(\mathbf{k}_i)^\top \mathbf{v}_i}{\phi(\mathbf{q}_t) \cdot \sum_{i=1}^t \phi(\mathbf{k}_i)^\top}.
\end{aligned} \tag{27}$$

In this way, each query $\mathbf{q}_t$ is multiplied with the aggregated key–value history, instead of being multiplied with each $\mathbf{k}_i$ separately, reducing the computational complexity from quadratic to linear.

Feature Map Design. Although the exponential kernel admits an exact feature map in theory, that map is infinite-dimensional, so a practical feature map $\phi(\cdot)$ must trade exactness for a finite dimension, and designing it requires careful consideration. To ensure efficient and effective linearized attention, a theoretically sound feature map should possess the following key attributes: (a) its output is non-negative; (b) its output dimension is finite; (c) when it is meant to mimic softmax attention, its approximation error to the exponential kernel is small. In Table 3, we present the representative choices of these feature maps. For instance, Katharopoulos et al. [219] choose the exponential linear unit [64] as the feature map. Random Feature Attention (RFA) [356] employs random Fourier features to build an unbiased estimator of the exponential kernel on ℓ_2 -normalized queries and keys; the variance of this estimator decreases as $1/D$ and is further reduced by the normalization and by a learned feature variance, but RFA gives no bound on the resulting error of the attention output. Performer [58] approximates the exponential kernel with feature maps built from positive orthogonal random features, but a large feature dimension is required to guarantee a small approximation error. Cosformer [363] stabilizes the linearized attention process by introducing the Fourier representation of the relative positions into the ReLU feature map. PolySketchFormer [214] proceeds in two steps: it first replaces the exponential kernel by a high-degree polynomial kernel $(\mathbf{q}_t \mathbf{k}_i^\top)^p$ with $p \geq 4$, which it shows empirically to match softmax quality, and then applies polynomial sketching to obtain a low-dimensional feature map whose inner product provably approximates this polynomial kernel; the approximation guarantee is thus with respect to polynomial attention,

not to softmax. Hedgehog [559] restores two properties of softmax attention weights that fixed feature maps lack: low-entropy spikiness, which lets each token concentrate on a few relevant tokens, and monotonicity in the query–key dot product, whose absence the authors hypothesize causes conflicting gradients when a pretrained Transformer is converted to linear attention. Its feature map is a learnable single-layer MLP with an element-wise exponential, applied to the queries and keys. When converting a pretrained model, the MLP is trained by attention-weight distillation, minimizing the cross-entropy between the linear attention distribution and the softmax attention distribution of the original layer, so the target is the normalized softmax weights rather than the raw dot products; when training from scratch, the distillation loss is dropped and the feature maps are learned end-to-end with the task loss. Based [12] uses a second-order Taylor approximation of the softmax kernel and combines the resulting global linear attention with sliding-window attention to capture both long-range recall and precise local dependencies. ReBased [4] replaces this fixed Taylor feature map with separately learned normalized feature maps for queries and keys. Element-wise squaring preserves non-negative similarities while removing the fixed lower bound of Based, allowing irrelevant tokens to be more strongly suppressed.

The above studies attempt to minimize the performance gap between linearized attention and dot product attention from a theoretical perspective. However, in practice, many simple functions (e.g., the linear function) are chosen as the kernels for convenience [455, 539, 285]. We will discuss these works in the following section.

6.5 Unification and Comparison of Linear Operators

This section investigates the *linear operators*, i.e., internal state update and output operators of linear language models, to uncover the strengths and potential of the corresponding models. We begin by analyzing the hidden state update formulas, as they form the core foundation of these models. This analysis identifies their common traits and differences within a unified framework. We further analyze these operators from other perspectives, such as expressivity and test-time training. Through this structured evaluation, we seek to offer insights that can guide the design of next-generation large language models.

Although recent language models employ linear operators proposed with different motivations, such as linearizing attention, linearizing RNNs, or using FWPs (discussed in Secs. 6.1 to 6.4), these operators share a remarkably similar mathematical form. Here we follow the general form summarized by Yang et al. [540] with more detailed notations to provide a comprehensive, unified framework for these linear operators:

$$\begin{aligned}\mathbf{H}_t &= \Gamma(\mathbf{x}_t) \odot \mathbf{H}_{t-1} + k(\mathbf{x}_t) \otimes v(\mathbf{x}_t), \\ z_t &= \gamma(\mathbf{x}_t) \cdot z_{t-1} + k(\mathbf{x}_t), \\ \mathbf{y}_t &= a(\mathbf{x}_t) \odot (q(\mathbf{x}_t, z_t) \cdot \mathbf{H}_t) + b(\mathbf{x}_t, z_t).\end{aligned}\tag{28}$$

Here, $\mathbf{H}_t$ stores the recurrent memory and z_t is the optional normalization state. For outer-product memories, $\mathbf{H}_t$ has d_k rows and d_v columns corresponding to the key and value dimensions, respectively; vector-state models are included by taking one of these dimensions to be 1. The functions k and v specify the outer-product write, q specifies the readout, and a and b provide multiplicative and additive output terms. The gates $\Gamma(\mathbf{x})$ and $\gamma(\mathbf{x})$ control how the memory and normalization states are carried forward, respectively, defining their state transitions [131]. The multiplication operator $\odot$ specifies how $\Gamma(\mathbf{x})$ acts on the entries of the old state $\mathbf{H}_{t-1}$. With element-wise multiplication, each entry can receive its own decay factor, providing fine-grained control over which features persist and for how long, as in Mamba [136]. Under matrix multiplication, a diagonal Γ shares each row’s decay across its d_v value entries, while a scalar multiple of the identity shares one factor across the entire head state. This prevents value-specific retention within a row, trading some gating flexibility for computational reuse [539, 70]. A non-diagonal Γ instead mixes entries along the key dimension within each value column, supporting key-directed memory correction [541] and, with suitable parameterizations, stronger state tracking without excessive arithmetic or storage overhead for state updates and parallel composition [354].

Some works simplify the linear operators by removing the normalization term z_t and the output gating term $a(\mathbf{x}_t)$ of Eq. (28), leading to a form which essentially belongs to the Unnormalized Linear Transformer (ULTRA) [406] family:

$$\begin{aligned}\mathbf{H}_t &= \Gamma(\mathbf{x}_t) \odot \mathbf{H}_{t-1} + k(\mathbf{x}_t) \otimes v(\mathbf{x}_t), \\ \mathbf{y}_t &= q(\mathbf{x}_t) \cdot \mathbf{H}_t + b(\mathbf{x}_t).\end{aligned}\tag{29}$$

By specifying the instances in Eq. (28) (the running normalizer z_t is omitted in the ULTRA family, Eq. (29)), the existing representative linear language models can be summarized in Table 4, grouped by the algebraic type of $\Gamma(\mathbf{x})$. The computation graph of Eq. (28) is depicted in Figure 11f.

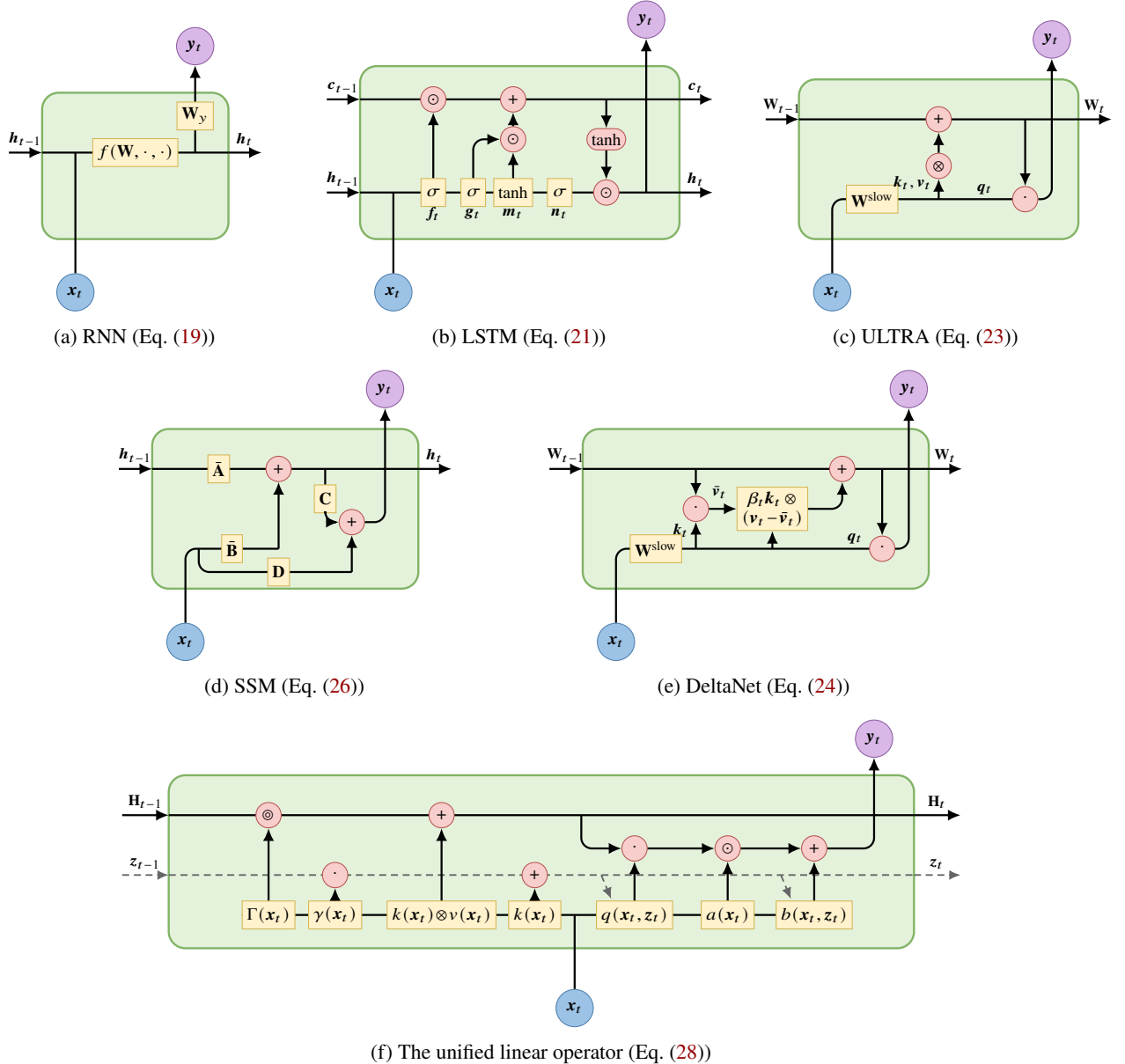


Figure 11: Schematic computation graphs of representative linear operators and their unification, drawn as unrolled recurrent cells. Each green cell performs the computation of one timestep: the blue circle is the input token x_t , the purple circle is the output y_t , and the horizontal black lines carry the recurrent states, entering from the previous timestep on the left and leaving toward the next timestep on the right. Yellow boxes denote learned transformations and gates, while pink nodes denote element-wise or matrix-product operations. (a)–(e): a vanilla RNN (Eq. (19)), LSTM [117], ULTRA [406], a discretized SSM [436, 136], and DeltaNet [403], where $\bar{v}_t = \phi(k_t)W_{t-1}$ retrieves the value currently associated with the key k_t before the error-correcting update. While the recurrences of the RNN and the LSTM are nonlinear in h_{t-1} (in the LSTM, h_{t-1} re-enters the gate activations f_t, g_t, m_t, n_t), the operators in (c)–(e) update their (matrix-valued) states linearly, enabling parallel training. (f): the unified linear operator of Eq. (28), which subsumes (c)–(e) and the models in Table 4: the bottom-row boxes are the gates and input-dependent functions, and the dashed line is the normalization state z_t , which is omitted in the ULTRA family (Eq. (29)).

	Hidden State Update				Output		
	$\mathbf{H}_t = \Gamma(\mathbf{x}_t) \odot \mathbf{H}_{t-1} + k(\mathbf{x}_t) \otimes v(\mathbf{x}_t), \quad \mathbf{z}_t = \gamma(\mathbf{x}_t) \cdot \mathbf{z}_{t-1} + k(\mathbf{x}_t)$				$\mathbf{y}_t = a(\mathbf{x}_t) \odot (q(\mathbf{x}_t, \mathbf{z}_t) \cdot \mathbf{H}_t) + b(\mathbf{x}_t, \mathbf{z}_t)$		
Model	$\Gamma(\mathbf{x}) \odot$	$k(\mathbf{x})$	$v(\mathbf{x})$	$\gamma(\mathbf{x})$	$q(\mathbf{x}, \mathbf{z})$	$a(\mathbf{x})$	$b(\mathbf{x}, \mathbf{z})$
Identity / cumulative: preserves $\mathbf{H}_{t-1}$ while new writes accumulate							
ULTRA [406]	$\mathbf{I} \cdot$	$\mathbf{xW}_K$	$\mathbf{xW}_V$	–	$\frac{\mathbf{xW}_Q}{\phi(\mathbf{xW}_Q)\mathbf{z}^\top}$	1	0
Linear Attention [488, 219]	$\mathbf{I} \cdot$	$\phi(\mathbf{xW}_K)$	$\mathbf{xW}_V$	1	$\frac{\phi(\mathbf{xW}_Q)}{\phi(\mathbf{xW}_Q)\mathbf{z}^\top}$	1	0
Scalar decay: scales all entries of $\mathbf{H}_{t-1}$ by one factor							
Gated RFA [356]	$\alpha_x \cdot$	$(1 - \alpha_x) \phi(\mathbf{xW}_K)$	$\mathbf{xW}_V$	α_x	$\frac{\phi(\mathbf{xW}_Q)}{\phi(\mathbf{xW}_Q)\mathbf{z}^\top}$	1	0
RetNet [455]	$\alpha \cdot$	$\mathbf{xW}_K$	$\mathbf{xW}_V$	–	$\frac{\mathbf{xW}_Q}{\phi(\mathbf{xW}_Q)\mathbf{z}^\top}$	1	0
mLSTM [24]	$\alpha_x \cdot$	$\beta_x \mathbf{xW}_K$	$\mathbf{xW}_V$	α_x	$\frac{\mathbf{xW}_Q}{\max\{1, \mathbf{xW}_Q\mathbf{z}^\top \}}$	$\sigma(\mathbf{xW}_A)$	0
Mamba-2 [70]	$\exp(-\Delta_x A) \cdot$	$\mathbf{xW}_K$	$\Delta_x \mathbf{x}$	–	$\mathbf{xW}_Q$	1	$\mathbf{x} \odot \mathbf{w}_D$
Diagonal / element-wise: scales entries of $\mathbf{H}_{t-1}$ without mixing them							
DFW [285]	$\alpha_{1,x} \otimes \alpha_{2,x} \odot$	$\mathbf{xW}_K$	$\mathbf{xW}_V$	–	$\mathbf{xW}_Q$	1	0
LRU [343]	$\alpha \odot$	$\mathbf{xW}_V$	1	–	$\mathbf{W}_Q$	1	$\mathbf{xW}_D$
GLA [539]	$\text{Diag}(\alpha_x) \cdot$	$\mathbf{xW}_K$	$\mathbf{xW}_V$	–	$\mathbf{xW}_Q$	1	0
Mamba [136]	$\exp(-[\text{Diag}(\Delta_x) \mathbf{A}]^\top) \odot$	$\mathbf{xW}_K$	$\Delta_x \odot \mathbf{x}$	–	$\mathbf{xW}_Q$	1	$\mathbf{x} \odot \mathbf{w}_D$
RWKV-6 [353]	$\text{Diag}(\alpha_x) \cdot$	$\mathbf{xW}_K$	$\mathbf{xW}_V$	–	$\mathbf{xW}_Q \Gamma^{-1}$	1	$[q \odot (\alpha_x \odot \mathbf{u} - 1)] k^\top v$
HGRN-2 [365]	$\text{Diag}(\alpha_x) \cdot$	$\mathbf{1} - \alpha_x$	$\mathbf{xW}_V$	–	$\phi(\mathbf{xW}_Q)$	1	0
RG-LRU [72]	$\alpha_x \odot$	1	$\sqrt{1 - \alpha_x^2} \odot \beta_x \odot \mathbf{x}$	–	1	1	0
Longhorn [271]	$\mathbf{1} - (k \odot k) \otimes \beta_x \odot$	$\mathbf{xW}_K$	$\beta_x \odot \mathbf{x}$	–	$\mathbf{xW}_Q$	1	$\mathbf{x} \odot \mathbf{w}_D$
MetaLA [60]	$\text{Diag}(\alpha_x) \cdot$	$\mathbf{1} - \alpha_x$	$\mathbf{xW}_V$	–	$\mathbf{xW}_Q$	1	$\sigma((q \odot k) \mathbf{w}^\top v)$
Delta-rule variants: mix entries of $\mathbf{H}_{t-1}$ through key-directed corrections							
DeltaNet [403]	$\mathbf{I} - \beta_x k \otimes k \cdot$	$\phi(\mathbf{xW}_K)$	$\beta_x \mathbf{xW}_V$	–	$\phi(\mathbf{xW}_Q)$	1	0
GDN [541]	$\alpha_x (\mathbf{I} - \beta_x k \otimes k) \cdot$	$\phi(\mathbf{xW}_K)$	$\beta_x \mathbf{xW}_V$	–	$\phi(\mathbf{xW}_Q)$	1	0
RWKV-7 [354]	$\text{Diag}(\alpha_x) - (\kappa_x \odot \beta_x) \otimes \kappa_x \cdot$	$\mathbf{xW}_K$	$\mathbf{xW}_V$	–	$\mathbf{xW}_Q$	1	$(q \odot k) \mathbf{w}^\top v$

Table 4: Representative linear operators under Eq. (28), grouped by the operation applied to $\mathbf{H}_{t-1}$ through $\Gamma(\mathbf{x}_t) \odot \mathbf{H}_{t-1}$.

Transition classes: Element-wise multiplication and multiplication by a diagonal matrix both rescale $\mathbf{H}_{t-1}$ without mixing its entries. Delta-rule variants instead use key-directed reads of $\mathbf{H}_{t-1}$ to construct corrections, optionally combined with decay. **Notation:** $\mathbf{x}$ denotes the single-head row-vector input, and subscript $\mathbf{x}$ marks input dependence: α_x is decay, β_x write strength, Δ_x the SSM step size, and κ_x a removal key. RetNet and LRU use input-independent α . Within each row, $q = q(\mathbf{x}, \mathbf{z})$, $k = k(\mathbf{x})$, $v = v(\mathbf{x})$, and $\Gamma = \Gamma(\mathbf{x})$ refer to the corresponding column entries. ϕ is a feature map or normalization, and σ the sigmoid. “–” means no normalizer $\mathbf{z}_t$; 1 in k, v, q, a denotes an absent factor. **Parameters:** $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$: query/key/value projections; $\mathbf{W}_A$: output-gate projection; $\mathbf{W}_D$ or $\mathbf{w}_D$: skip connection; $\mathbf{A}$ (scalar A): positive SSM state parameter; $\mathbf{u}$: bonus vector; $\mathbf{w}$: learned vector. **Scope:** Model-specific gate parameterizations, scalings, biases, and input preprocessing (e.g., convolutions) are omitted; see the cited papers.

As grouped in the table, the most notable differences among linear operators stem from the algebraic type of the gate $\Gamma(\mathbf{x})$: identity accumulation, scalar decay, diagonal or channel-wise decay, or a rank-1 / generalized-Householder overwrite. The majority of these linear operators eliminate the need for the normalization term $\mathbf{z}_t$. Unlike dot-product attention mechanisms [219], which rely on softmax normalization, many linear variants, including RetNet [455] and GLA [539], omit this term to simplify their formulations. Furthermore, the design of the key $k(\mathbf{x})$ and value $v(\mathbf{x})$ functions varies significantly across models. Linear Attention [219] and RetNet [455] employ straightforward linear transformations for simplicity, whereas models like Gated DeltaNet [541] as well as RWKV-7 [354] incorporate more complex functions (e.g., write-strength gating, token shift, and value residuals, whose details are left to the original papers) to enhance expressiveness and adaptability. Some models also introduce sophisticated scaling terms $a(\mathbf{x})$ and bias terms $b(\mathbf{x})$ to the output.

Comparison and Analysis. We classify the expressive power of linear recurrent/attention models based on the following perspectives in Table 5: (a) from the perspective of the gating mechanism $\Gamma(\mathbf{x}) \odot$, we analyze it according

Method	Expressivity			Capacity	Adaptability		
	Diagonal/ GH/Scalar Gating	Data-Independent Gating	Non-negative Spectrum	State Expansion	Test-time Optimization	Test-time Adaptive Step Size	Test-time Weight Decay
ULTRA [406] / Linear Attention [488, 219]	—	—	—		Analytic		
Gated RFA [356]	✓		✓		Analytic		
LRU [343]	✓	✓			Analytic		
RG-LRU [72]	✓		✓		Analytic		
RetNet [455]	✓	✓	✓	✓	Analytic		
DFW [285]	✓		✓	✓	Analytic		
RWKV-6 [353]	✓		✓	✓	Analytic		
GLA [539]	✓		✓	✓	Analytic		
HGRN-2 [365]	✓		✓	✓	Analytic		
MetaLA [60]	✓		✓	✓	Analytic		
Mamba [136]			✓	✓	Analytic		
Mamba-2 [70]	✓		✓	✓	Analytic		
mLSTM [24]	✓		✓	✓	Analytic		
DeltaNet [403]	✓		✓	✓	GD	✓	
Longhorn [271]			✓	✓	Analytic	✓	
Gated DeltaNet [541]	✓		✓	✓	GD	✓	✓
RWKV-7 [354]				✓	GD	✓	✓

Table 5: We categorize the linear language models of Table 4 (ULTRA and Linear Attention share one row) according to expressivity restrictions in their gating mechanisms, state dimension expansion [537], and the test-time regression view [498]. GH denotes a generalized Householder matrix (identity minus a scaled vector outer product), and non-negative spectrum means that all eigenvalues of the effective recurrent transition are non-negative real values. “—” denotes that no gating variable is present.

to three aspects: its format⁵, its data dependency, and its value range; (b) the expansion of state dimension [537] greatly enlarges the capacity of hidden states; (c) Recent works [541, 498] adopt the perspective of test-time training (TTT) [454, 26, 271, 491] to provide another view for these operators. For linear networks, this TTT formulation can be equivalently cast as the FWP update [408], where the model weights are updated during the forward pass to act as a dynamic associative memory. Sun et al. [454] instantiate this idea as concrete sequence layers: TTT-Linear and TTT-MLP use a linear model and an MLP as their recurrent states, respectively, and update them by gradient descent on a learned self-supervised objective. Subsequent methods refine this test-time update. Titans augments mini-batch gradient descent with data-dependent momentum and forgetting, allowing recent loss gradients to influence subsequent updates while controlling memory retention [26]. MesaNet replaces the single gradient step on the latest key–value pair with an approximate solution to a regularized, forget-weighted regression over all pairs observed up to the current step [491]. At a more general level, test-time regression [498] provides a unified form for these attention operators. It separates two choices: a per-step regression objective and the solver applied to it. We use the following generalized objective for the hidden-state update of linear operators:

$$\mathcal{L}_t(\mathbf{H}) = \frac{1}{2} \sum_{i=s_t}^t \omega_i^{(t)} \|\mathbf{v}(\mathbf{x}_i) - \mathbf{k}(\mathbf{x}_i)\mathbf{H}\|_2^2 + \frac{\lambda_t}{2} \|\mathbf{H}\|_F^2 + \frac{\delta_t}{2} \|\mathbf{H} - \mathbf{H}_{t-1}\|_F^2, \quad \mathbf{H}_t = \text{Solve}(\mathcal{L}_t, \mathbf{H}_{t-1}). \quad (30)$$

In Eq. (30), s_t and the nonnegative weights $\omega_i^{(t)}$ select the historical key–value associations to be fitted, while the terms weighted by λ_t and δ_t penalize a large state and a departure from $\mathbf{H}_{t-1}$, respectively. The objective alone does not fix the recurrence: Solve denotes the solver applied at each step, which may be exact minimization of $\mathcal{L}_t$, a single explicit gradient step from zero or from $\mathbf{H}_{t-1}$, or an approximation of the exact minimizer. A single gradient step is in general not the minimizer, so the same objective yields different operators under different solvers, and the operators below are identified by both choices. For the unregularized cumulative case with $s_t = 1$ and $\lambda_t = \delta_t = 0$, choosing $\omega_i^{(t)} = \prod_{j=i+1}^t \eta_j$, where each $\eta_j \in [0, 1]$ is a forgetting factor, makes earlier associations undergo more forgetting. The

⁵Note that while the gating variable $\Gamma(\mathbf{x})$ in some models may not strictly belong to the specific forms, e.g., diagonal/generalized-Householder/scalar forms, the gating mechanism can degenerate into the forms that exhibit reduced capacities during unfolding [294, 131], such as Mamba.

following examples show how different settings of $(s_t, \omega^{(t)}, \lambda_t, \delta_t)$ and different solvers in Eq. (30) recover familiar linear operators.

Linear Attention is recovered by setting $s_t = 1$, $\omega_i^{(t)} = 1$, and $\lambda_t = \delta_t = 0$ and taking one unit-step batch gradient update from a zero candidate state matrix (or equivalently, by approximating the key Gram matrix as the identity in the analytical least-squares solution), whereas DeltaNet is recovered by setting $s_t = t$, $\omega_i^{(t)} = 1$, and $\lambda_t = \delta_t = 0$ and taking one online gradient update from $\mathbf{H}_{t-1}$ [219, 403, 498]. The accumulator obtained in this way is the unnormalized form of Eq. (29); the running normalizer $\mathbf{z}_t$ of Eq. (28) (the denominator of Eq. (27)) is not part of the regression state and has to be carried separately when a normalized readout is wanted. Keeping $s_t = t$, $\omega_i^{(t)} = 1$, and $\delta_t = 0$ but taking $\lambda_t > 0$, an online gradient step of size ρ_t gives $\mathbf{H}_t = (1 - \rho_t \lambda_t) \mathbf{H}_{t-1} + \rho_t \mathbf{k}_t \otimes (\mathbf{v}_t - \mathbf{k}_t \mathbf{H}_{t-1})$. In this update, the ℓ_2 term produces the overall decay factor $1 - \rho_t \lambda_t$ on the old state, whereas the gradient of the prediction loss produces the error-correction term associated with the current key. With $\alpha_t = 1 - \rho_t \lambda_t$, $\beta_t = \rho_t / \alpha_t$, and $\tilde{\mathbf{v}}_t = \alpha_t \mathbf{v}_t$, the same update can be written exactly as $\mathbf{H}_t = \alpha_t (\mathbf{I} - \beta_t \mathbf{k}_t \otimes \mathbf{k}_t) \mathbf{H}_{t-1} + \beta_t \mathbf{k}_t \otimes \tilde{\mathbf{v}}_t$, which is the Gated DeltaNet recurrence [541, 498]. For the multiplicative weights defined above, the exact least-squares solution contains the inverse of $\sum_{i=1}^t \omega_i^{(t)} \mathbf{k}(\mathbf{x}_i) \otimes \mathbf{k}(\mathbf{x}_i)$ when this matrix is invertible. Approximating this matrix by the identity yields $\mathbf{H}_t \approx \eta_t \mathbf{H}_{t-1} + \mathbf{k}_t \otimes \mathbf{v}_t$, a linear attention whose old state is decayed by the scalar gate η_t [498]. This is the scalar-gated special case of Gated Linear Attention, corresponding to the scalar-decay rows of Table 4; the per-key-dimension gate $\text{Diag}(\alpha_t)$ of GLA proper [539] does not follow from a scalar weight $\omega_i^{(t)}$, because it requires the forgetting to act separately on each key dimension rather than on each association as a whole. Finally, when $s_t = t$, $\lambda_t = 0$, and $\delta_t = 1$, exact minimization with scalar $\omega_i^{(t)}$ gives $\mathbf{H}_t = \mathbf{H}_{t-1} + \frac{\omega_t^{(t)}}{1 + \omega_t^{(t)} \|\mathbf{k}_t\|_2^2} \mathbf{k}_t \otimes (\mathbf{v}_t - \mathbf{k}_t \mathbf{H}_{t-1})$. The factor multiplying the prediction-error correction shows that its strength is determined by the current association weight and key norm rather than by a fixed learning rate. Rearranged, this exact (proximal) update reads $\mathbf{H}_t = (\mathbf{I} - \varepsilon_t \mathbf{k}_t \otimes \mathbf{k}_t) \mathbf{H}_{t-1} + \varepsilon_t \mathbf{k}_t \otimes \mathbf{v}_t$ with $\varepsilon_t = \omega_t^{(t)} / (1 + \omega_t^{(t)} \|\mathbf{k}_t\|_2^2)$, i.e., a rank-one state mixing along the current key.

6.6 Future Directions

Linear recurrent models have attracted substantial interest because they replace quadratic attention over the sequence length with recurrent updates that admit parallel training. This efficiency is a central advantage of large language models for context scaling, particularly for agents that must complete long-horizon tasks. At the same time, several challenges and opportunities remain:

- **State Capacity:** an immediate constraint for linear recurrent models arises from the fixed capacity of the recurrent state. As the context grows, linear models must compress an increasing amount of historical information into a representation whose size remains unchanged, which may compromise both information fidelity and direct retrieval. Some researchers have also reported that linear variants can still underperform full attention on certain tasks in large-scale experiments [122].
- **Richer Transitions and Hybridization:** Several responses to these limitations are already in progress. One direction is to enrich the state transition. For example, fully input-dependent non-diagonal transitions let state channels route information to one another and can represent permutation composition while remaining parallelizable through a scan, although dense transition products introduce additional computational and optimization costs [294]. Beyond enriching the transition within each layer, depth provides another source of expressivity: Heo et al. [159] show that stacking restricted parallelizable SSM layers can represent increasingly complex order-sensitive dynamics. A practical response is to interleave linear and softmax attention, often at a 3:1 ratio, as in Qwen3-Next and Qwen3.5/3.6 [477, 372, 373], and Kimi Linear and Kimi K3 [471, 317]. This design has already scaled beyond earlier experiments: Kimi K3 uses a 2.8T-parameter model with 104B active parameters and a 1M-token context [317].
- **Efficient Kernels:** Supporting these evolving architectures also requires continued progress in efficient implementations. Training and prefill typically use chunkwise-parallel kernels, which pass a fixed-size state between chunks and compute token interactions within each chunk in parallel [455, 539, 364, 538, 249, 477, 372]. Autoregressive decoding instead uses fused recurrent kernels to update the state for one or a few new tokens [538, 477, 372]. Efficient chunk kernels combine algebraic reformulation with explicit GPU-kernel blocking and operator fusion. For delta-rule recurrences, one such reformulation collects the token-by-token state updates within a chunk and rewrites them as a lower-triangular solve followed by matrix

multiplications. GPU-kernel implementations then divide this work into small blocks that fit in registers and shared memory and fuse adjacent operations, reducing intermediate writes to HBM [538, 471]. For long-context prefill, implementations gain additional parallelism by processing multiple token blocks from each chunk concurrently [25] and overlapping within-chunk computation with state propagation between chunks [317]. Sequence segments can also be processed independently within one GPU or across GPUs and then combined to recover exact boundary states [317]. Kimi K3 further modifies KDA by limiting the maximum decay at each token. This keeps temporary scale factors within the BF16 range, so all causal blocks can use the same dense matrix-multiplication path [317].

- **New State Organizations and Operators:** Other exploratory directions change the organization of the recurrent state or reuse recurrent mechanisms in other operators. Recent work retains linear recurrence while combining it with hierarchical or routed memory structures to increase capacity [140, 85]. Related extensions also adapt recurrent operators to multidimensional inputs [366]. A deeper understanding of the structural connections between softmax attention and DeltaNet, as well as the mathematical properties of DeltaNet, may also inform future operator design [447, 449, 448]. Related work also applies gating, cumulative forgetting, and delta-style corrections to softmax attention while retaining normalized retrieval [368, 265, 577].

7 Mixture of Experts

The development of large language model architectures has been significantly enhanced by Mixture of Experts [36], a key technique that offers a way to scale the capacity of the model without proportionally increasing computational costs. The core concept of MoE, i.e., a modularized network guided by a learnable router, dates back to the late 1980s [144], with subsequent work introducing hierarchical structures [200, 211]. Their application in modern language models [434] has led to breakthroughs in both model scaling and efficiency.

The key innovation of modern MoE lies in its sparse activation pattern. Rather than processing all inputs through the same set of parameters, MoE strategically employs a gating mechanism to route tokens to specialized expert neural networks [434]. Both the gating mechanism and the specialized experts are trained at the same time. Hence the expert FLOPs per token are determined by the number and width of the active experts rather than by the total number of experts n . The decoupling is not complete, however: a dense router still scores all n experts, which is negligible for tens or hundreds of experts but is the reason that million-expert designs resort to product-key routing (Sec. 7.1), and parameter memory, optimizer state, expert dispatch, and inter-device communication all grow with n rather than with the number of active experts. This selective activation enables models to grow to enormous sizes—reaching even trillions of parameters [96]—while keeping computational requirements manageable during both training and inference.

Formally, an MoE layer [434] consists of n experts $E_1, \dots, E_n : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$ (in Transformers, typically FFNs) and a routing function $r : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^n$. For an input $\mathbf{x}$, the output is a router-weighted sum of expert outputs:

$$\mathbf{y} = \sum_{i \in \chi} r(\mathbf{x})_i \cdot E_i(\mathbf{x}),$$

where only the experts in the router-selected set χ are evaluated; their number $k = |\chi|$ is a hyperparameter, typically small, whereas n can reach thousands or even millions [153].

The remainder of this section traces three overlapping developments (Sec. 7.1): the sparsely gated FFN that first made this mapping trainable at scale, along with the load-balancing and stability techniques it required; the fine-grained shared-expert layout that became the open-weight default; and the systems-oriented variants that reshape experts under communication and memory constraints. Table 6 lists selected milestones along these lines.

7.1 Evolution of MoE in LLMs

The following three developments summarize the evolution of MoE in LLMs; their histories overlap rather than forming a strictly sequential progression.

The Era of Sparse Gating. This period established the default recipe: replace selected feed-forward layers with a pool of large experts and let a learned token-choice router activate a small top- k subset. Its design questions were, in turn, how to define the layer and its router, how to keep the router from collapsing onto a few experts, how to fit the resulting variable load onto fixed hardware, and where in the network to place the layers.

Category	Model	Key Contribution
Sparse Gating	Sparsely-Gated MoE [434]	Introduces the first large-scale sparse MoE layer (top- k gating) with auxiliary load balancing losses.
	GShard [246]	Scales top-2 gated MoE Transformers to 600B parameters with a single differentiable load-balancing loss and expert capacity limits.
	Switch [96]	Demonstrates efficacy of routing to a single expert; introduces selective precision.
	BASE [248]	Formulates token routing as a linear assignment problem to maximize efficiency.
	Hash [389]	Eliminates learnable gating parameters by employing static hash-based routing.
	Expert Choice [579]	Inverts routing: experts select top- k tokens to ensure load balance.
	ST-MoE [590]	Analyzes training instability and introduces router Z-loss for stabilization.
	SMEAR [322]	Proposes soft merging of expert weights to enable efficient dense MoE training.
	DS-MoE [345]	Combines dense training gradients with sparse inference for robust optimization.
Fine-Grained + Shared	DeepSpeed-MoE [382]	Introduces pyramid-style architectures and non-uniform expert distribution.
	DeepSeekMoE [68]	Implements fine-grained expert segmentation and shared expert isolation.
	Scaling Laws [62, 236]	Derives scaling laws for dense versus sparse models and for expert granularity.
	Monet [348]	Scales expert counts up to 262K per layer toward near-monosemantic specialists.
Systems-Oriented	MoE++ [210]	Extends the expert pool with zero-computation experts such as copy or zero.
	ScMoE [37]	Shortcut-connects routed experts so that expert communication can overlap with the dense path.
	LatentMoE [90]	Routes and computes in a compressed latent space to reduce all-to-all traffic and expert weight loads.

Table 6: Selected milestones in Mixture of Experts research, grouped by the three overlapping developments in Sec. 7.1.

Layer and Router. Large-scale sparsely gated MoE was demonstrated with LSTM language models by Shazeer et al. [434], and was subsequently integrated into Transformer FFNs in work such as GShard [246]. These Transformer architectures replace the feed-forward network (FFN) in selected blocks with MoE layers. This choice comes from several compelling observations about feed-forward blocks. First, they are the most parameter-heavy part of the Transformer block. Furthermore, they exhibit natural sparsity in their activations [569, 260, 588]. Additionally, the inherent independence of FFN computation makes it particularly amenable to expert-based parallelization, as demonstrated in Switch Transformers [96]. Furthermore, scaling laws have been developed for language models with MoE layers: Clark et al. [62] model the loss as a function of the dense parameter count and the number of experts. Open-weight models such as Mixtral [206] later popularized this coarse, token-choice recipe as a standard production layout.

In standard learned routing, the routing function is a trainable module that maps each input token to a set of real-valued scores or probabilities over the experts. Specifically, given an input $\mathbf{x}$, the router computes a weight vector $r(\mathbf{x}) \in \mathbb{R}^n$ (often via a linear transform followed by a softmax and optional noise injection), selects the top- k entries, and dispatches $\mathbf{x}$ to the corresponding experts for processing, thereby enabling conditional computation.

Load Balancing. A critical challenge in training MoE models is the so-called expert collapse, in which the routing function converges to favor a small number of experts, causing the remaining experts to receive few or no input tokens and effectively become inactive; the phenomenon was documented in early conditional-computation work [88] and addressed explicitly by Shazeer et al. [434]. This collapse often arises from positive feedback loops during training: experts that receive more tokens receive more gradient updates and improve faster, which leads the router to allocate even more tokens to them. The resulting underutilization of diverse experts negates the benefits of conditional computation.

Two families of remedies have been proposed. The first, and most popular, adds an auxiliary loss that penalizes polarized distributions of selected experts [434, 246]; the second changes the routing rule so that balance holds by construction. The original load balancing mitigation [434] used two losses: L_{imp} , which pushed the sum of the weights of the experts to be uniform across the batch, and L_{lb} , which encouraged the uniform distribution of tokens across experts. This was simplified by GShard [246] and later popularized in Switch Transformer [96]. The simplified version uses a single simple load balancing loss. Let X be the batch and $T = |X|$ be the total number of tokens in a batch, then

$$L_{\text{aux}}(X) = \alpha n \sum_{i=1}^n C_i(X) P_i(X)$$

where

- $C_i(X) = \frac{1}{T} \sum_{\mathbf{x} \in X} \mathbf{1}_{\{i \in \text{top-}k(r(\mathbf{x}))\}}$ is the fraction of tokens assigned to expert i in a given batch
- $P_i(X) = \frac{1}{T} \sum_{\mathbf{x} \in X} r(\mathbf{x})_i$ is the average router probability assigned to expert i over the batch. Unlike C_i , it is differentiable with respect to the router parameters, which is what allows the loss to provide a gradient signal.

With exactly k assignments per token and normalized router probabilities $r(\mathbf{x})$, $\sum_i C_i(X) = k$ and $\sum_i P_i(X) = 1$. $C_i(X)$ is non-differentiable as it relies on the discrete top- k selection. Consequently, the gradient propagates only through the term $P_i(X)$. In this optimization landscape, $C_i(X)$ effectively acts as a weighting coefficient for $P_i(X)$. Minimizing the product $\sum C_i P_i$ penalizes the router for assigning high probability mass to experts that are already frequently selected (high C_i), thereby encouraging a more uniform distribution of probability mass across all experts.

σ -MoE [66] shows that when the sigmoid function is used as a router gate as proposed in [248], the auxiliary losses can be simplified by maximizing the entropy of the distribution instead. Sigmoid scores weight expert outputs, while the regularizer first averages the softmax probabilities of the router logits across all batch tokens, $\bar{p} = |X|^{-1} \sum_{\mathbf{x} \in X} \text{softmax}(s(\mathbf{x}))$, where $s(\mathbf{x}) \in \mathbb{R}^n$ denotes the router logits, and then maximizes the entropy of $\bar{p}$. It does not maximize each token’s entropy independently. On the other hand, AdaMoE [553] proposes scheduling the weight of load balancing loss, such that the load balancing is the strongest at the beginning of the training and diminished at the end; they observed improved performance with such a training scheme.

ReMoE [505] replaces the top- k softmax router with a ReLU router, so that the number of active experts per token is determined by which router outputs are positive; sparsity is controlled by an adaptive ℓ_1 regularizer, and weighting this regularizer by per-expert activation rates plays the role of a load-balancing term.

The second family enforces balanced expert loads by construction rather than through a soft penalty. BASE layers [248] treat the routing as a constrained linear assignment problem during training, so that every expert receives the same number of tokens. Expert-Choice Routing [579] flips the problem upside down and makes the experts choose their tokens, which balances the load per expert but lets the number of experts per token vary. A third mechanism, Loss-Free Balancing [499], removes the auxiliary loss without imposing a hard constraint: it adds a per-expert bias to the routing scores used for the top- k selection (not to the weights applied to expert outputs) and, after each training step, moves the bias of every expert by a fixed amount in the direction that corrects its load on the previous batch. Balance is thus maintained by feedback over training steps and measured empirically, rather than guaranteed within a batch as in the assignment-based methods.

Capacity and Router Stability. Load balancing shapes the routing distribution in expectation, but for individual batches, tokens could still cluster heavily non-uniformly. This intersects with hardware constraints in practical deployments. Modern GPUs have finite memory and computational capacity that cannot be exceeded. Without a hard capacity limit, individual experts could receive more tokens than they can physically process, leading to out-of-memory errors or computational bottlenecks.

The standard way to deal with this practical issue involves defining a capacity factor [246, 96] κ , which controls the maximum number of tokens any expert can reach:

$$\text{tokens_per_expert} = \frac{T \cdot k}{n} \cdot \kappa$$

When an expert receives more assignments than its capacity, overflow assignments may bypass the expert computation through the residual path. A simple positional admission policy drops assignments later in the flattened token order. Batch Prioritized Routing [387] instead prioritizes tokens by router scores, while Random Token Selection [226] randomizes their admission priority; these policies determine which assignments fit within capacity. Separately, Switch Transformers studied No-Token-Left-Behind, which reroutes overflow tokens to alternative experts, but reported no empirical benefit [96]. Ignoring capacity rounding, $\kappa = 1$ requires perfectly uniform expert loads if all Tk assignments are to be processed without dropping; the capacity factor itself does not enforce this balance. With $\kappa > 1$, extra capacity permits load variation at a higher memory cost.

Besides load, the router can also be numerically unstable. The router z-loss introduced in ST-MoE [590] is a way to stabilize router training. It penalizes large logits entering the softmax operation to prevent numerical instabilities caused

by the exponentiation mechanism:

$$L_z(\mathbf{S}) = \alpha_z \cdot \frac{1}{T} \sum_{t=1}^T \left(\log \left(\sum_{i=1}^n e^{s_{t,i}} \right) \right)^2$$

where T is the number of tokens and n denotes the number of experts. $\mathbf{S} \in \mathbb{R}^{T \times n}$ collects the logits entering the router, with $s_{t,i}$ denoting the logit for token t and expert i . Here, α_z is a hyperparameter controlling the regularization strength. The router z-loss drives the log-partition function of the router logits toward zero, which keeps the logits small in magnitude and the exponentiation numerically well-conditioned (a concern in reduced-precision training). It does not by itself constrain the gaps between logits: a sharply peaked routing distribution can have zero z-loss, so its benefit is the empirically observed reduction of training instabilities rather than a guarantee against extreme gating decisions. The authors of ST-MoE compare it against other stabilizers, including update clipping in the optimizer, and find that while both improve training stability, clipping degrades model quality whereas the z-loss does not. The total loss is then a sum of the prediction loss, the load balancing loss, and the z-loss.

Placement. The placement and frequency of MoE layers throughout the network architecture significantly impact both performance and efficiency. Early large-scale MoE Transformers placed MoE layers in every second layer, a pattern established by GShard [246] and followed by GLaM [86] and DeepSpeed-MoE [382], while ST-MoE [590] opted for a sparser distribution with MoE layers appearing once every four layers. DeepSpeed-MoE [382] studied the non-uniform distribution of experts and observed that it is preferable to have experts in the second half layers over the first half. The intuition behind this is that later layers are usually more specialized than the more general layers in the first half. Most recent open-weight MoE LLMs have since moved to denser integration, incorporating MoE layers in every Transformer block [206, 68, 320], often with the exception of the first one or a few layers, which are kept dense for stability.

Replacement of FFNs remains dominant. However, researchers have explored applying MoE-style routing to attention components. Mixture of Attention heads (MoA) [563] demonstrates the benefits of using MoE in the attention layer. SwitchHead [67] improves the mechanism to reduce the number of necessary matrix computations. Mixture of Sparse Attention (MoSA) [359] implements sparse attention as an expert choice routing mechanism for attention heads, so that each head processes only the tokens routed to it; its effect on the KV cache is discussed in Sec. 5.1. Routing has also been applied at the granularity of whole blocks: Mixture-of-Depths (MoD) [384] treats the entire block and the “empty” residual connection as the two routing options, allowing the model to dynamically allocate compute across tokens (see also Sec. 13). Widely deployed MoE LLMs such as DeepSeekMoE [68] and Mixtral [206] nevertheless keep their attention layers dense and apply expert routing only to the FFN.

The Fine-Grained Shared-Expert Paradigm. The second development, introduced by DeepSeekMoE [68], reshapes the expert pool itself along two axes: a few experts are made *shared* and always active, and the routed experts are made *fine-grained*, that is, smaller and more numerous.

Shared Experts. A small number of experts are always active for every token. The intent is that they capture knowledge common to all tokens, so that the routed experts are freed to specialize and redundancy among them is reduced; DeepSeekMoE supports this with ablations in which isolating one shared expert improves most benchmarks at equal parameters and compute, replacing the shared expert by one more routed expert raises the Pile loss from 1.808 to 2.414, and disabling the top routed experts hurts DeepSeekMoE more than GShard, indicating lower redundancy among its routed experts [68]. This shared-plus-routed layout has since been adopted by DeepSeek-V2/V3 [73, 269] and many other open-weight MoE LLMs (see Table 14).

Fine-Grained Experts. Although the MoE layer typically contains a modest number of large feed-forward experts, of which one or two are selected per token, fine-grained MoEs [68, 236] spread the capacity over many small experts. Rather than allocating each token to one of a few monolithic experts, fine-grained designs subdivide the layer into a large number of small experts and activate several of them per token, which increases the number of distinct expert combinations available at a fixed FLOP budget; whether individual micro-experts specialize in narrowly defined functions is an empirical question that has been studied only in specific settings [236]. Krajewski et al. [236] extend the earlier dense-versus-sparse scaling laws to expert granularity and the training-token budget, and fine-grained expert configurations can improve the FLOPs-to-performance trade-off in the settings studied by DeepSeekMoE and Krajewski et al. [68, 236].

Ultra-Sparse Experts as Memory. This line of work has revealed a close connection between MoEs and Product-Key Memory Networks (PKMs) [241, 66], where expert selection can be viewed as sparse memory addressing. PKM-style

routing mechanisms make it possible to efficiently index and activate extremely large expert pools, as demonstrated by PEER [153], which addresses over a million tiny experts using product-key retrieval and exhibits superior scaling behavior compared to dense and coarse-grained MoEs. Similarly, Monet [348] scales expert counts up to 262K per layer to induce highly specialized, near-monosemantic experts, and reports interpretability and targeted-editing results for this expert design. This perspective also connects MoEs to memory networks: the soft addressing in end-to-end memory networks [451] provides a general analogy, while sparse memory retrieval offers a closer counterpart to sparse expert selection [29, 179, 180]. Task-specific variants such as MoWE [84] further demonstrate that tying microexperts to individual vocabulary items effectively constructs large implicit memory stores that excel on knowledge-intensive tasks. Together with recent empirical scaling-law studies [236], these results suggest that increasing expert count while shrinking expert size is not merely an implementation detail, but a principled pathway toward more efficient, interpretable, and scalable model architectures.

Independently Trained Experts. In certain scenarios, experts are pre-trained independently on different datasets and combined in a later stage into a single MoE. This approach has been explored in several recent MoE models [251, 452, 437]. Gerstenberger et al. [118] additionally proposed to distill a large mixture of independently pre-trained models into a single compact model, following the spirit of Irie et al. [187].

Scale-Driven and Systems-Oriented Variants. The third development relaxes two assumptions of the standard recipe once models reach production scale: that all experts are identical, and that the expert layer is a self-contained block whose communication and memory costs are taken as given.

Heterogeneous Experts. Although most practical implementations use identical neural network architectures for all experts within a layer to simplify implementation and hardware utilization, recent research explores heterogeneous experts with varying architectures or capacities. This heterogeneity can potentially capture diverse patterns in data better than homogeneous experts. Variations include HMoE [493], which combines experts of different sizes, and MoE++ [210], which incorporates zero-computation experts such as copying or zeroing an input. A related form of heterogeneity operates at the block level rather than among experts: Arctic [440] combines a dense residual Transformer with a parallel MoE branch of homogeneous experts, and other works treat skip connections as experts [384].

Communication- and Memory-Aware Designs. As MoE models have moved into production at the hundred-billion to trillion-parameter scale, architectural choices are increasingly driven by systems bottlenecks, chiefly the all-to-all communication of expert parallelism and the memory bandwidth consumed by loading expert weights. Shortcut-connected MoE (ScMoE) [37] restructures the block so that, via a shortcut connection, the routed experts of a layer process the representation of an earlier position in the block rather than the immediately preceding output. This decouples expert communication from its conventional sequential ordering and allows it to be overlapped, up to entirely, with the computation of the dense path. LongCat-Flash [474] adopts this shortcut-connected design at the 560B scale, combining it with zero-computation experts so that the compute spent per token can vary with token difficulty. Along the complementary axis of memory and communication volume, LatentMoE [90] projects each token from the hidden dimension into a smaller latent space in which both the routing payload and the expert computation reside, reducing expert weight loads and all-to-all traffic proportionally to the compression ratio; the savings are reinvested into a larger total number of experts and a higher top- k , improving accuracy per FLOP and per parameter at a similar serving cost. This latent-expert design has been adopted by the Nemotron 3 Super and Ultra models [334, 335], and a similar latent MoE scheme is used in Kimi K3 [317] (see Table 14). Orthogonal to routing, the memory footprint of the expert weights themselves can be reduced after training; MC-MoE [178], for instance, applies expert-aware adaptive quantization and pruning for this purpose.

7.2 Future Directions

Despite rapid progress, several aspects of MoE design remain open:

- **Routing Mechanisms:** The diversity of routing mechanisms in recent MoE architectures suggests that there is still potential in improving the routing mechanism. Current approaches span from standard token-based routing [434] to expert choice routing [579], linear programming formulations [248], hashing [389], or uniform sampling [593]. Each method brings its own balance of computational speed, routing quality, and training stability, but the continuous emergence of new methods suggests there is still room for improvement. Future

work could explore the combination of the strengths of existing approaches or the development of entirely new routing paradigms.

- **Expert Placement across Layers:** The optimal distribution of experts across different layers of deep neural networks remains an open question that warrants further investigation. Although current architectures often employ a uniform distribution of experts across layers as in [246], some work has been done in this direction. As discussed in Sec. 7.1, the PR-MoE design of DeepSpeed-MoE [382] puts more experts in the later layers. Brainformers [580] derive a stackable block found with evolutionary search that opposes the standard uniform distribution of experts. However, there is still limited understanding of how expert placement affects model performance and computational efficiency.
- **Heterogeneous Experts:** Heterogeneous experts, i.e., experts with different architectures, capacities, or specialized structural biases (see the early examples in Sec. 7.1), remain underexplored. While heterogeneous experts offer promising directions for specialization, they introduce significant systems-level challenges. Current efficient training and inference systems heavily rely on the homogeneity of experts for parallelization. Developing hardware-aware routing mechanisms and general systems that can efficiently train and serve heterogeneous experts remains an open problem.

8 Activation Functions

Activation functions are a cornerstone of neural network design, serving as the primary source of non-linearity that enables models to learn complex patterns beyond simple linear relationships [142]. In the context of LLMs, which are predominantly based on the Transformer architecture, the choice of activation function within the position-wise Feed-Forward Network (FFN) sub-layers has a profound impact on training dynamics, model performance, and computational efficiency [433]. While the multi-head attention mechanism is often credited with the Transformer’s success in capturing long-range dependencies, the FFN layers are increasingly understood to be the primary repositories of the model’s parametric knowledge [119]. Consequently, the activation function is not a mere implementation detail but a critical component that governs how this vast store of knowledge is accessed, processed, and transformed at each layer of the network.

8.1 Evolution of Activation Functions in LLMs

The following four developments summarize the evolution of activation functions in LLMs; their histories overlap rather than forming a strictly sequential progression. Table 7 compares the formulations and properties of the most common choices.

The Era of Rectification. This period was dominated by the Rectified Linear Unit (ReLU) [105, 106] and its direct descendants. ReLU is computationally simple and was used in the original Transformer proposed by Vaswani et al. [489]. Its shortcomings, however, notably the “dying ReLU” problem, spurred the development of numerous variants designed to introduce non-zero gradients for negative inputs. Besides ReLU, a family of variants was proposed, each aiming to introduce a small, non-zero gradient for negative inputs to keep neurons alive and learning. Leaky ReLU (LReLU) [284] was one of the earliest and simplest modifications, defined as $f(x) = \max(\alpha x, x)$, where α is a small, fixed hyperparameter, such as 0.01. For positive inputs, it behaves identically to ReLU, but for negative inputs, it has a small, non-zero slope of α . This “leak” ensures that neurons can recover even if their inputs become negative, as they will still receive a gradient signal. Parametric ReLU (PReLU) [151] extends the idea of Leaky ReLU by making the slope α a learnable parameter rather than a fixed hyperparameter. This allows the network to adaptively learn the optimal leakage slope for each neuron or channel during training, providing greater flexibility. Exponential Linear Unit (ELU) [64] offers a more sophisticated solution, defined as $f(x) = x$ if $x > 0$ and $f(x) = \alpha(e^x - 1)$ if $x \leq 0$. Like LReLU and PReLU, ELU provides a non-zero gradient for negative inputs. However, it has the additional advantage that for negative inputs, the function saturates to a negative value, which can help push the mean of the neuron activations closer to zero. This zero-centering property can accelerate learning by addressing the biased gradient issue of standard ReLU.

Scaled Exponential Linear Unit (SELU) [231] has self-normalizing properties, pulling neuron activations toward a fixed point with zero mean and unit variance. This can theoretically eliminate the need for external normalization while ensuring stable gradient flow in the training of deep networks. However, the mathematical foundations of SELU rely on

specific weight initializations and feed-forward architectures, which may be incompatible with the dot-product attention mechanisms used in modern LLMs.

The Transition to Smooth, Probabilistic Activations. Following the era of rectification, the research community began to explore activation functions that moved beyond the piecewise-linear and non-smooth nature of ReLU and its variants. This led to the development of a new class of smooth, continuously differentiable functions, most notably the Gaussian Error Linear Unit (GELU) [156], Sigmoid Linear Unit (SiLU), introduced in the GELU paper and independently studied for reinforcement learning [156, 92], and Swish [383]. GELU quickly gained prominence due to its strong empirical performance and elegant theoretical motivation. It is mathematically formulated as: $\text{GELU}(x) = x\Phi(x)$, where $\Phi(x)$ is the cumulative distribution function (CDF) of the standard normal distribution. The core theoretical insight behind GELU is its connection to stochastic regularizers like Dropout [146, 160, 442, 103, 102]. The authors framed GELU as a way to bridge the gap between deterministic non-linearities (like ReLU) and stochastic regularization techniques. GELU can be interpreted as the expected value of a stochastic gating transformation that either keeps the input (the identity map) or sets it to zero. Specifically, the input is kept with probability $\Phi(x)$ and dropped otherwise, where Φ denotes the standard normal CDF. Around the same time as GELU’s rise, Ramachandran et al. proposed another activation function named Swish [383]. Swish is defined by the remarkably simple formula: $\text{Swish}(x) = x\sigma(\beta x)$, where $\sigma(\cdot)$ is the standard sigmoid function and β may be fixed or learned. At $\beta = 1$, Swish is identical to SiLU [156, 92, 383]. The design of Swish [383] was inspired by the gating mechanisms used in recurrent architectures [117], where a sigmoid function acts as a gate to control information flow.

The Modern Paradigm of Gated Linear Units. The evolution from smooth, self-gated functions like GELU [156], SiLU [92] and Swish [383] to the explicit gating mechanisms of the Gated Linear Unit (GLU) [117, 71] family represents the most significant architectural shift in the FFN block since the modern quadratic Transformer. GLU adapted the multiplicative gating mechanism from LSTMs [117] for use in feed-forward networks. This paradigm, which introduces two parallel computational paths for content and gating, allows for a more powerful and dynamic form of information control. The seminal work [433] systematically demonstrated that integrating GLU variants into the Transformer FFN yields substantial performance improvements. This research was not merely about swapping one activation function for another; it introduced a third weight matrix, with a narrower intermediate dimension used to match the parameter and computation budgets of a two-matrix FFN. The robust empirical success of one particular variant, SwiGLU [433], has led to a remarkable convergence in the field, with multiple, independent, state-of-the-art LLMs from leading research labs adopting it as their standard [30, 295, 20]. This widespread consensus underscores the profound impact of GLU-based activations and solidifies their position as the current state of the art.

Scale-Driven Revivals and Stability-Oriented Variants. Despite the dominance of SwiGLU, the demands of large-scale training have recently motivated two pragmatic departures from this consensus. The first is the revival of Squared ReLU (SqReLU), defined as $f(x) = \max(0, x)^2$, which was selected through evolutionary architecture search in Primer [441]. SqReLU retains the sparsity-inducing rectification of ReLU while its quadratic positive branch provides a stronger, higher-order non-linearity; moreover, unlike GLU variants, it requires only two rather than three FFN weight matrices, freeing the parameter budget for a wider intermediate dimension. These properties have led to its adoption in the hybrid Mamba-Transformer Nemotron-H and Nemotron 3 families [331, 330]. The second departure addresses numerical robustness rather than expressiveness: since the linear branches of SwiGLU are unbounded, rare but extreme activation values can exceed the limited dynamic range of the low-precision formats used in modern training and inference. Clamped SwiGLU limits these pre-activations asymmetrically. In the GPT-OSS reference implementation and MiniMax-M3 configuration [339, 337, 305], the gate branch is capped above at 7, while the value (up) branch is clipped to $[-7, 7]$. With $\bar{g} = \min(g, 7)$ and $\bar{u} = \text{clip}(u, -7, 7)$, the intermediate output is $\bar{g} \sigma(1.702\bar{g}) \odot (\bar{u} + 1)$, followed by the output projection. Frontier models also continue to introduce new gated units of their own, such as the SiTU-GLU channel-mixing layer of Kimi K3 [317] (see Table 14). Both trends illustrate that at the current scale of LLMs, the choice of activation function is increasingly co-determined by systems considerations, such as parameter allocation and numerical precision, rather than by task accuracy alone.

8.2 Future Directions

The widespread adoption of SwiGLU [433] reflects strong empirical results for this gated FFN design. However, the emergence of scale-dependent challenges and the renewed interest in the efficiency of sparse activations suggest that the

Function / gated layer	Formulation	Properties	Key Advantage
ReLU [105]	$\max(0, x)$	Non-smooth, Monotonic	Computationally cheap, induces sparsity
GELU [156]	$x\Phi(x)$	Smooth, Non-monotonic	Smooth ReLU surrogate, probabilistic motivation
SiLU [156, 92]	$x\sigma(x)$	Smooth, Non-monotonic	Empirically strong performance, self-gating
Swish [383]	$x\sigma(\beta x)$	Smooth, Non-monotonic	Empirically strong performance, self-gating
Squared ReLU [441]	$\max(0, x)^2$	C^1 , Monotonic	Sparse like ReLU, higher-order non-linearity
GEGLU [433]	$(x\mathbf{V}) \odot \text{GELU}(x\mathbf{W})$	Smooth, Non-monotonic	Strong empirical performance, dynamic gating
SwiGLU [433]	$(x\mathbf{V}) \odot \text{SiLU}(x\mathbf{W})$	Smooth, Non-monotonic	SOTA performance, widely adopted
Clamped SwiGLU [339]	$(\bar{u} + 1) \odot \bar{g} \sigma(1.702 \bar{g})$	Continuous, bounded, piecewise smooth	Robust in low-precision formats

Table 7: Comparison of scalar activation functions and GLU-family gated layers. Scalar nonlinearities and the Hadamard product act element-wise; the GLU rows also include learned matrix projections. The final FFN output projection is omitted from these rows. Here $\sigma(\cdot)$ is the sigmoid function and $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. For GLU-family layers, $\mathbf{W}$ and $\mathbf{V}$ denote the weight matrices of the gate and value linear projections, respectively, and the bias terms of these projections are optional. For Clamped SwiGLU, $\bar{g} = \min(x\mathbf{W}, 7)$ and $\bar{u} = \text{clip}(x\mathbf{V}, -7, 7)$ are the clamped gate and value branches; the clamps introduce derivative discontinuities at the clipping thresholds, so the layer is continuous but not smooth, and “bounded” refers to its output before the omitted output projection.

search is far from over. The future of non-linearity in LLMs is unlikely to be a “one-size-fits-all” solution. Instead, we are likely moving towards a more heterogeneous and adaptive landscape. Future research may focus on:

- **Task- and Layer-Specific Activations:** Models may employ different activation functions in different layers or for different tasks, potentially learned automatically through methods like evolutionary search.
- **Hardware-Aware Design:** The choice of activation function will be increasingly co-designed with hardware accelerators and numerical formats in mind, balancing peak theoretical performance with practical deployment constraints.
- **Principled Discovery:** As our theoretical understanding of deep learning matures, new activation functions may be derived from first principles, such as information theory or dynamical systems, moving beyond purely empirical or heuristic-driven design.
- **Application in Other Modules:** This RNN-era gating idea [167, 166] has not only flourished in Transformer FFNs as gated activations, but could also be applied to other parts of the architecture. For example, Self-Adjust Softmax [576] uses gating to modulate the Softmax output so that strict normalization is not required, increasing expressiveness, while Gated Attention [368] applies gating to the attention mechanism to improve the distribution of attention outputs. These works illustrate attempts to extend gated-activation ideas beyond FFNs to other modules (see also the discussion of Gated Attention in Sec. 9.1).

9 Normalization and Residual Connections

In addition to fundamental components such as attention mechanisms and feed-forward networks within the Transformer architecture, two critical elements that significantly enhance the performance and training stability of Transformers are normalization and residual connections. Normalization techniques, such as layer normalization [13], rescale the activations entering or leaving each sub-layer, which improves the conditioning of optimization. Residual connections [444, 152], whose additive identity path was anticipated across time by the constant error carousel of LSTM [167], facilitate the gradient flow through the network, effectively mitigating challenges such as vanishing gradients in training very deep neural networks [443, 152]. In particular, gated skip connections across depth were introduced by Highway Networks [444]; the ungated special case popularized by ResNets [152] is what is now incorporated into every sub-layer of modern Transformer architectures.

9.1 Normalization Methods

BatchNorm is an influential normalization method [183], which, although effective in certain contexts, posed challenges for sequential data due to its dependence on batch statistics. This limitation spurred the development of alternative methods that were better suited for language tasks. LayerNorm [13] addresses the limitations of BatchNorm by normalizing across features within a single data sample. For a Transformer, it computes statistics across the features of each token. Before the learned affine transformation, the normalized features have zero mean and variance close to one, with the exact variance affected by the numerical stabilizer ϵ . RMSNorm [555] simplifies LayerNorm by omitting the mean subtraction step. It normalizes activations by their root mean square (RMS) value, reducing computational overhead while maintaining performance. RMSNorm has been particularly effective in training deep Transformer models, offering a balance between simplicity and efficacy. Non-Parametric LayerNorm, studied by Xu et al. [526] under the name LayerNorm-simple and adopted by the first OLMo models [132], modifies the standard LayerNorm by removing the learnable gain and bias, focusing solely on normalizing the activations without scaling or shifting. Xu et al. found that the learnable parameters contribute little to LayerNorm’s benefit and can increase overfitting, so this simplification reduces parameters and computation at little cost, although it removes the model’s ability to rescale normalized features per dimension. As shown in Table 8, these methods have been widely applied to popular LLMs.

Apart from the above techniques that are used in recent LLMs, there are alternative approaches aimed at optimizing language modeling performance. CRMSNorm [207] introduces a compression mechanism that enhances efficiency while retaining the advantages of RMSNorm, thereby reducing the computational burden and making it particularly suitable for large-scale language models. WeightNorm [395] reparameterizes weight vectors to decouple their magnitude from their direction. This technique accelerates convergence by controlling the norm of the weights, resulting in a more stable and efficient training process. ScaleNorm [328] normalizes activation vectors by their ℓ_2 norm and then scales them by a learned parameter. SphereNorm [273] is a batch-agnostic normalization method that constrains features to a unit hypersphere via angular projection. Its normalization does not depend on batch statistics, avoiding this source of sensitivity to batch size. This geometric approach inherently bounds outputs and effectively mitigates covariate shift in deep networks. These aforementioned methods offer additional perspectives on achieving training stability while simultaneously enhancing performance.

Beyond normalizing the inputs or outputs of entire sublayers, normalization can also be applied *inside* the attention module to directly control the scale of attention logits. QK-Norm [158] normalizes the query and key vectors immediately before their dot product, so that each logit becomes a cosine similarity multiplied by a learnable scale; this removes the uncontrolled growth of logits with the query and key norms and makes the softmax less prone to saturating into low-entropy, near one-hot distributions, without ruling them out, since the learnable scale (or the gain of a LayerNorm/RMSNorm variant) can still make the bounded logits large. Originally proposed with ℓ_2 normalization and a learnable temperature to stabilize low-resource machine translation [158], the technique was revived at scale when applying LayerNorm to queries and keys was found to eliminate the uncontrolled growth of attention logits that destabilized the training of ViT-22B [78], an instability later reproduced and analyzed systematically in small-scale proxy studies [511]. In modern LLMs, the variant that applies RMSNorm to queries and keys per head has become a common stability component, adopted by OLMo 2 [336], Qwen3 [371], Gemma 3 [126], and GLM-4.5 [479], among others; an asymmetric variant that normalizes only the keys (K-Norm) has also been explored [181]. Since the normalization discards the norms of individual queries and keys (a loss of information that the original paper argues does not sacrifice expressivity), the practical trade-off is a constraint on the attention logits in exchange for a substantially enlarged stable region of training hyperparameters. A related stabilizer acts on the attention *output* rather than on its logits: Gated Attention [368] applies a query-dependent, head-specific sigmoid gate to the output of scaled dot-product attention. The gate inserts a non-linearity between two otherwise consecutive linear maps and yields input-dependent sparsity, which the authors show removes attention sinks and improves training stability; it has been adopted by Qwen3-Next [477] (see Table 14).

9.2 Combinations of Normalizations and Residual Connections

The integration of normalization techniques and residual connections is pivotal in the architecture of LLMs. The specific arrangement of these components significantly influences the training dynamics and overall model performance [523]. A critical aspect to consider is the “seesaw effect” between gradient vanishing and representation collapse. This effect highlights the delicate balance required: while certain configurations may mitigate gradient vanishing, they can

Method	Formula	Related Models
LayerNorm [13]	$\gamma \cdot (\mathbf{x} - \mu) / \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2 + \epsilon} + \beta$	GPT-3 [34], BLOOM [510]
Non-Parametric LayerNorm [132]	$(\mathbf{x} - \mu) / \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2 + \epsilon}$	OLMo [132]
RMSNorm [555]	$\gamma \cdot \mathbf{x} / \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2 + \epsilon} + \beta$	LLaMA [486], Gemma [467], Qwen [20]
QK-Norm [158]	$\text{softmax}(g \cdot \text{Norm}(\mathbf{q}) \text{Norm}(\mathbf{k})^\top)$	OLMo 2 [336], Qwen3 [371], Gemma 3 [126]
CRMSNorm [207]	$\mathbf{x} / \sqrt{(\sum_{i=1}^{d-1} x_i^2 + (\sum_{i=1}^{d-1} x_i)^2) / d + \epsilon}$	–
WeightNorm [395]	$g \cdot \mathbf{w} / \ \mathbf{w}\ $, $\mathbf{w}$ represents the weight vector	–
ScaleNorm [328]	$g \cdot \mathbf{x} / \ \mathbf{x}\ $, $\mathbf{x}$ represents the input vector	–
SphereNorm [273]	$g \cdot (\mathbf{x} \mathbf{w}^\top) / (\ \mathbf{x}\ \cdot \ \mathbf{w}\)$	–

Table 8: Different normalization methods. $\mathbf{x}$ denotes the input vector, d its dimension (the hidden width), and μ the mean of $\mathbf{x}$. The parameters γ and β are learnable vectors and g is a learnable scalar, while ϵ is a small constant for numerical stability. For QK-Norm, $\mathbf{q}$ and $\mathbf{k}$ denote a query and a key vector, $\text{Norm}(\cdot)$ is an ℓ_2 , Layer, or RMS normalization applied per head, and g plays the role of a learnable temperature. CRMSNorm operates on the $(d - 1)$ -dimensional compressed representation of a zero-mean d -dimensional vector, which is why its sums run to $d - 1$ while the average is taken over d . In the context of WeightNorm, $\mathbf{w}$ denotes the weight vector. In the context of SphereNorm, $\mathbf{w}$ and $\mathbf{x}$ denote the weight and input vector, respectively.

inadvertently lead to representation collapse, and vice versa [583, 584, 503]. Achieving an optimal configuration is essential for effective and efficient model training.

The original Transformer architecture [489] utilized the Post-LayerNorm (*Post-LN*) strategy, where layer normalization is applied subsequent to the residual connections. Although this approach proved effective in certain scenarios, it becomes unstable as depth grows: Xiong et al. [523] showed that at initialization the gradients of the parameters near the output are large under Post-LN, which is why a learning-rate warm-up stage is required, and Wang et al. [496] attribute the instability of deep Post-LN models to exploding model updates. The Pre-LayerNorm (*Pre-LN*) configuration, which applies layer normalization to the input of each sub-layer so that the residual stream itself is never normalized, had already been adopted in practice by GPT-2 [379], the Sparse Transformer [56], and deep NMT models [500] when Xiong et al. [523] provided a theoretical analysis of it, showing that Pre-LN keeps the gradient scale well behaved at initialization and can therefore be trained without warm-up and with larger learning rates.

DeepNorm [496], a stabilized variant of Post-LN, was subsequently proposed to enable the training of extremely deep Transformers. It achieves this by a simple yet effective architectural modification: scaling the residual connection by a factor $\alpha > 1$ and down-scaling the initial weights by $\beta < 1$. Under its analysis (SGD updates with the proposed α and β), this bounds the magnitude of the *model update*, $\|F(\mathbf{x}; \theta^*) - F(\mathbf{x}; \theta)\|$, by a constant; it is the change of the network’s output, not the parameter update itself, that is controlled. DeepNorm thereby combines the strong performance of Post-LN with the training stability of Pre-LN.

To further augment training stability, the Sandwich-LN [81] technique was introduced, which incorporates an additional layer normalization operation at the end of each residual branch, effectively “sandwiching” the sub-layer between two normalization operations. CogView reports that the additional normalization helps stabilize large-scale training [81]. Another notable modification is the “norm-after” (residual post-norm) strategy adopted by the OLMo-2 model [336], following earlier use in Swin Transformer V2 [276] and Chameleon [42]. It normalizes the *output* of each attention and feed-forward sub-layer before it is added to the residual stream, i.e., $\mathbf{x} + \text{Norm}(f(\mathbf{x}))$; unlike Post-LN, the residual stream itself is never normalized, and unlike Pre-LN, the contribution of each sub-layer is bounded. Experimental results in OLMo-2 suggest that this arrangement reduces gradient-norm spikes, thereby enhancing training stability. Figure 12 presents a diagram illustrating these methods.

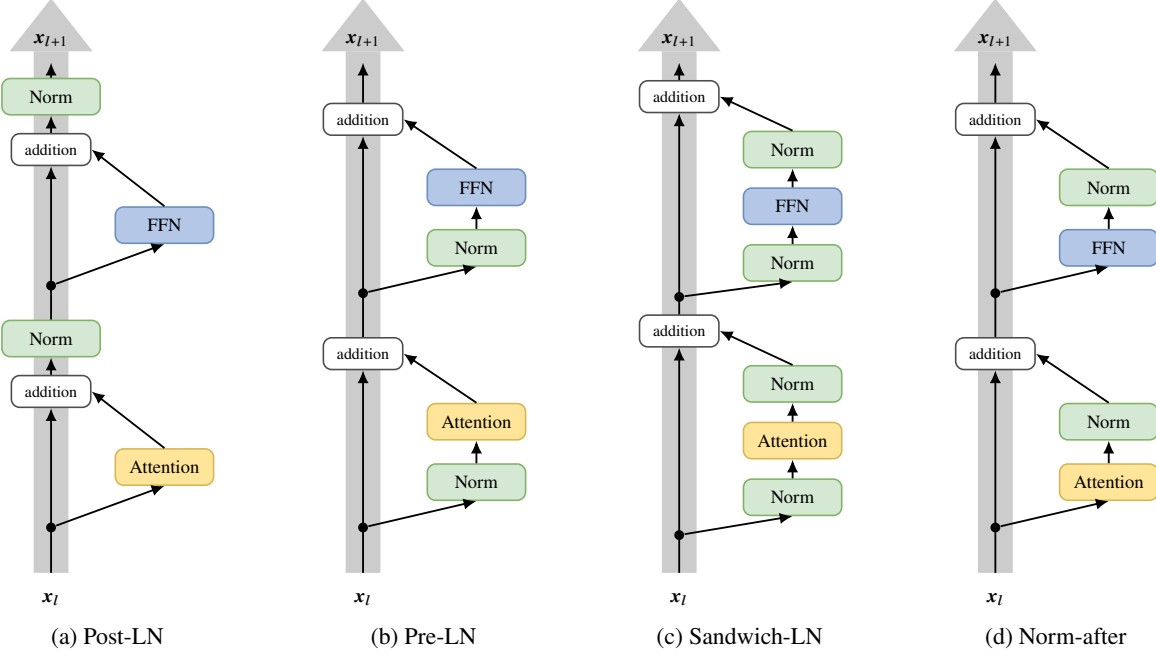


Figure 12: Different intra-block combinations of normalization and residual connection. The gray band is the residual stream carrying x_l to x_{l+1} , black dots mark where the stream is read by a sub-layer, and white boxes are the residual additions. (a) Post-LN normalizes the residual stream after each addition; (b) Pre-LN normalizes only the input of each sub-layer; (c) Sandwich-LN normalizes both the input and the output of each sub-layer; (d) norm-after normalizes the output of each sub-layer before it is added to the stream.

To further enhance the equilibrium between gradient flow and representation integrity, recent studies have introduced normalization and residual propagation strategies for Transformer architectures. The Mix-LN method [252] combines the strengths of Pre-LN and Post-LN by applying Post-LN in the earlier layers and transitioning to Pre-LN in the deeper layers. This configuration is reported to yield more uniform gradient norms across the network, thereby improving training stability and overall performance. Similarly, HybridNorm [589] integrates the advantages of both normalization approaches from an intra-layer perspective by employing Query-Key-Value (QKV) normalization within the attention mechanism and Post-LN in the feed-forward network of each Transformer block. This design not only stabilizes training but also enhances performance, particularly in the context of LLMs. Additionally, Hyper-Connections [583] offer an alternative to traditional residual connections, enabling the network to dynamically adjust the strength of connections between features at varying depths. This flexibility effectively addresses the limitations associated with fixed residual connections, thereby mitigating the seesaw effect and achieving a more balanced gradient flow while preserving representation richness. Based on Hyper-Connections [583], DeepSeek proposes Manifold-Constrained Hyper-Connections (mHC) [522], which stabilizes training of large-scale networks by constraining the learned mixing matrices of the residual streams to be doubly stochastic (i.e., to lie in the Birkhoff polytope), enforced via Sinkhorn normalization. An exactly doubly stochastic residual-mixing matrix A is a convex combination of permutation matrices and is non-expansive in the Euclidean norm: $\|A\|_2 \leq 1$. This bounds amplification by residual mixing, but does not imply norm preservation or an identity map, and does not bound the Jacobian of the entire nonlinear layer. Finite Sinkhorn iterations approximate the doubly stochastic constraint. In the reported experiments, mHC [522] reduces training instabilities relative to unconstrained Hyper-Connections [583]; it has been adopted in models such as DeepSeek-V4 [76], and Hy4 [483] likewise uses a multi-stream hyper-connection variant (iHC, with four residual streams; see Table 14). Attention Residuals (AttnRes) [472] generalize the residual connection in a different direction. The standard Pre-LN residual stream accumulates all preceding layer outputs with fixed unit weights, which causes uncontrolled hidden-state growth with depth and progressively dilutes the contribution of each individual layer. AttnRes replaces this fixed accumulation with softmax attention over the outputs of preceding layers, allowing each layer to selectively aggregate earlier representations with learned, input-dependent weights. To keep the memory and communication overhead tractable at scale, the block-wise variant Block AttnRes partitions layers into blocks and attends over block-level

representations, making the scheme a practical drop-in replacement for standard residual connections. Integrated into the Kimi Linear architecture [471], AttnRes yields more uniform output magnitudes and gradient distributions across depth, and it has since been adopted at scale by Kimi K3 [317].

9.3 Future Directions

The evolution from Post-LN [489] to Pre-LN [379, 523], and further to advanced configurations such as Mix-LN [252], Sandwich-LN [81], and Hyper-Connections [583], demonstrates the continuous efforts to address the inherent challenges of gradient vanishing and representation collapse. Each design innovation aims to strike a balance between maintaining stable gradient flow and preserving expressive representation capabilities, thus optimizing model performance across different scales and depths.

While recent approaches have made significant progress, there remain challenges and opportunities for further exploration:

- **Dynamic and Adaptive Normalization:** Future architectures could benefit from adaptive normalization mechanisms that adjust dynamically to the training stage or model depth. For instance, utilizing Mix-LN [252] principles with layer-wise learned hyperparameters might improve gradient stability and feature diversity.
- **Theoretical Insights into the Seesaw Effect:** Further theoretical studies are needed to model and quantify the seesaw effect more rigorously, providing insights into how normalization and residual connections interact at scale [583]. This could inform principled designs rather than relying solely on empirical tuning.
- **Beyond Layer Normalization:** Alternative normalization methods such as GroupNorm [514] and ScaleNorm [328], as well as normalization-free designs, could be revisited in the context of modern LLMs. Dynamic Tanh (DyT) [585] replaces every normalization layer by an element-wise $\tanh(\alpha x)$ with a learnable scale α , removing normalization as an operation. nGPT [281] takes the opposite route: it removes LayerNorm/RMSNorm but re-normalizes the hidden state to unit norm after every attention and MLP update and normalizes the embedding dimension of all weight matrices, so normalization is not eliminated but turned into parameter-free norm constraints applied throughout the residual stream. Whether either design retains its benefits at LLM scale remains an open question.

10 Positional Encoding

As introduced in Sec. 4.3, the Positional Encoding (PE) module supplements token representations with explicit position information, which the permutation-equivariant attention operation cannot otherwise access (causal masking breaks this symmetry and lets decoder-only models recover some positional information implicitly, as discussed in Sec. 10.4, but explicit PE remains the norm). Beyond this foundational role, the design of PE strongly influences several key dimensions of model capability, including length generalization (the capacity to process sequences longer than those observed during training), expressiveness (how position interacts with content, e.g., content–position disentanglement and content-adaptive positions), and architectural compatibility (integration with diverse Transformer variants and data modalities). These broader considerations have motivated a steady evolution of positional encoding strategies—from early absolute formulations to more advanced relative and hybrid approaches.

In this section, we trace this progression, beginning with Absolute Positional Encoding (APE) and advancing through Relative Positional Encoding (RPE) and subsequent innovations, highlighting how each paradigm addresses distinct facets of the positional encoding challenge.

10.1 Absolute Positional Encoding

The first solutions to the position problem involved Absolute Positional Encoding (APE), where each position in the sequence is mapped to a unique representation. This can be achieved through either a fixed, deterministic function or through learnable parameters. Table 9 summarizes the formulations of representative APE methods.

Sinusoidal APE. As introduced in Sec. 4.3, Vaswani et al. [489] introduced a fixed PE method that uses sine and cosine functions of varying frequencies to generate a unique positional vector for each position. By assigning different frequencies to each dimension of the embedding vector, this method produces a unique d -dimensional vector for each absolute position in the sequence.

Method	PE Formulation	Learnable	Integration
Learned [110, 378]	$\mathbf{e}_i^{\text{pos}} = \mathbf{W}_{\text{pos}}[i]$, the i -th row of a learnable matrix $\mathbf{W}_{\text{pos}}$	✓	Add
Sinusoidal [489]	$\mathbf{e}_{i,2j}^{\text{pos}} = \sin\left(\frac{i}{10000^{2j/d}}\right)$, $\mathbf{e}_{i,2j+1}^{\text{pos}} = \cos\left(\frac{i}{10000^{2j/d}}\right)$		Add
SHAPE [230]	$\mathbf{e}_{i,2j}^{\text{pos}} = \sin\left(\frac{i+k}{10000^{2j/d}}\right)$, $\mathbf{e}_{i,2j+1}^{\text{pos}} = \cos\left(\frac{i+k}{10000^{2j/d}}\right)$, $k \sim \mathcal{U}\{0, \dots, K\}$ per sequence		Add
FLOATER [275]	$\mathbf{e}_{i+1}^{\text{pos}} = \mathbf{e}_i^{\text{pos}} + \int_i^{i+1} h(\tau, \mathbf{e}^{\text{pos}}(\tau); \theta_h) d\tau$	✓	Add

Table 9: Comparison between different APEs, in the notation of Sec. 4.3: $\mathbf{e}_i^{\text{pos}} \in \mathbb{R}^d$ is the positional embedding of position i , d is the embedding dimension, and j indexes its dimension pairs $(2j, 2j + 1)$. k is the random offset that SHAPE [230] samples per training sequence, with the maximum shift K as its hyper-parameter, and $h(\cdot; \theta_h)$ is the learnable vector field of FLOATER [275], whose encodings are samples $\mathbf{e}_i^{\text{pos}} = \mathbf{e}^{\text{pos}}(i)$ of the continuous trajectory $d\mathbf{e}^{\text{pos}}(\tau)/d\tau = h(\tau, \mathbf{e}^{\text{pos}}(\tau); \theta_h)$; the increment between consecutive positions is the integral of h over the unit interval, not h evaluated at the start point.

This approach is deterministic and introduces no learned positional parameters. Its sinusoidal values remain bounded, and for any fixed offset k , the encoding at position $i + k$ is a linear transformation of the encoding at position i . The original Transformer paper [489] argued that this structure could facilitate learning relative-position relationships and extrapolation to longer sequences. The encoding can be evaluated beyond the training range, but this mathematical property alone does not guarantee that a trained model will maintain its quality at those lengths.

Learned APE. An alternative to the fixed sinusoidal function is to treat the positional embeddings as learnable parameters, analogous to how semantic word embeddings are learned. In this scheme, a lookup table or embedding matrix is created. Each row of this matrix corresponds to a positional vector that is initialized randomly and then updated via backpropagation during the model’s training process. This approach has been adopted by several prominent models, including BERT [80], GPT-2 [379], and GPT-3 [34].

The primary advantage of learnable APE is that the positional vectors can adapt to patterns in the training data. A finite learned lookup table, however, defines embeddings only up to its configured maximum position. Without an extension mechanism, it cannot provide vectors beyond that range; extending the table or interpolating its entries requires an additional design choice and does not by itself guarantee extrapolation quality.

Other APE Variants. To overcome the generalization issues of both fixed and learnable APEs, a family of methods has emerged that augments the encoding process during training. These techniques typically build upon the sinusoidal formulation but introduce forms of randomization to prevent the model from overfitting to absolute positional cues. SHAPE [230] introduces a simple yet effective augmentation. During each training step, all position indices within a sequence are shifted by a single, randomly sampled offset k . This prevents the model from memorizing that a specific feature always occurs at a fixed position (e.g., position 5) and instead encourages it to learn patterns based on relative distances between tokens, which remain invariant under the global shift. CAPE [264] applies a richer set of augmentations to continuous position values. In addition to a global shift similar to SHAPE, CAPE introduces a local shift, where each position is perturbed by a different random value, and global scaling, where all positions are multiplied by a random scalar. These combined augmentations further discourage the model from memorizing absolute positions or specific relative distances, promoting better generalization.

Another line of research reconceptualizes position not as a discrete index but as a continuous variable, aiming to generate positional representations that vary more smoothly. Complex-Valued Embeddings [495] extend the embedding space into the complex plane, modeling the embedding of a word as a continuous function of its position. Each coordinate of a word embedding is parameterized as $g(\text{pos}) = re^{i(\omega \cdot \text{pos} + \theta)}$ with word-specific amplitude r , frequency ω , and initial phase θ , so that the amplitude can encode semantic information while the phase component, which rotates with position pos , encodes positional information. This provides a principled mathematical framework for ensuring that word representations evolve smoothly as their position changes. FLOATER [275] models the sequence of positional encodings as the output of a continuous dynamical system, leveraging the framework of Neural Ordinary Differential Equations (ODEs). Instead of learning a static embedding for each position, FLOATER learns the dynamics of change—a “latent

Method	$\text{logits}_{i,j}$	Content–position interaction	Learnable	Remark
Transformer-XL [69]	$\mathbf{q}_i \mathbf{k}_j^\top + \mathbf{q}_i (\mathbf{r}_{i-j} \mathbf{W}_{k,r})^\top + \mathbf{u} \mathbf{k}_j^\top + \mathbf{v} (\mathbf{r}_{i-j} \mathbf{W}_{k,r})^\top$	✓	✓	$\mathbf{r}_{i-j}$: sinusoidal relative-position vector; $\mathbf{W}_{k,r}$: learnable positional projection; $\mathbf{u}, \mathbf{v}$: learnable global bias vectors
T5 Bias [381]	$\mathbf{q}_i \mathbf{k}_j^\top + b_{i-j}$		✓	b_{i-j} is a learnable scalar
ALiBi [361]	$\mathbf{q}_i \mathbf{k}_j^\top - r i-j $			r is a hyperparameter
KERPLE [54]	$\mathbf{q}_i \mathbf{k}_j^\top - r_1 \log(1 + r_2 i-j)$		✓	$r_1, r_2 > 0$ are learnable per-head kernel parameters
Sandwich [55]	$\mathbf{q}_i \mathbf{k}_j^\top + r \sum_{k=1}^{d'} \cos\left(\frac{i-j}{10000^{k/d'}}\right)$			r, d' are hyperparameters
FIRE [254]	$\mathbf{q}_i \mathbf{k}_j^\top + f_\eta\left(\frac{\psi(i-j)}{\psi(\max\{L, i\})}\right)$		✓	$\psi(x) = \log(cx + 1)$; f_η is a learnable function
CoPE [124]	$\mathbf{q}_i \mathbf{k}_j^\top + \mathbf{q}_i \mathbf{e}[p_{ij}]^\top$	✓	✓	$p_{ij} = \sum_{k=j}^i g_{ik}$, $g_{ik} = \sigma(\mathbf{q}_i \mathbf{k}_k^\top)$; $\mathbf{e}$ is a learnable embedding
DAPE [575]	$\mathbf{q}_i \mathbf{k}_j^\top + f_\eta(\mathbf{Q} \mathbf{K}^\top, \text{bias}(i, j))$	✓	✓	$\text{bias}(i, j)$: additive positional encoding; $\mathbf{Q}, \mathbf{K}$: query and key matrices
Inkling [484]	$\mathbf{q}_i \mathbf{k}_j^\top + \mathbb{1}[0 \leq i-j < L_r] \mathbf{r}_i \mathbf{P}_{:, i-j}$	✓	✓	$\mathbf{r}_i = \mathbf{x}_i \mathbf{W}_r$: input-conditioned feature; $\mathbf{P}$: bank of d_r learnable bias-vs-distance profiles; no positional term beyond L_r
RoPE [450]	$(\mathbf{q}_i \mathbf{R}_i^\top)(\mathbf{k}_j \mathbf{R}_j^\top)^\top = \mathbf{q}_i \mathbf{R}_{j-i} \mathbf{k}_j^\top$ $= \sum_{c=1}^{d/2} r_{q_{i,c}} r_{k_{j,c}} \cos((j-i)\theta_c + \phi_{k_{j,c}} - \phi_{q_{i,c}})$	✓		$\mathbf{R}_i$: position-dependent rotation matrix; $\mathbf{q}_{i,c}, \mathbf{k}_{j,c}$: c -th 2D components; r_*, ϕ_* : content magnitude and phase; θ_c : positional frequency
PoPE [127]	$\sum_{c'=1}^d r_{\tilde{q}_{i,c'}} r_{\tilde{k}_{j,c'}} \cos((j-i)\theta_{c'} + \delta_{c'})$	✓	✓	$\tilde{q}_{i,c'}, \tilde{k}_{j,c'}$: complex components; r_* : content-dependent magnitude; $\theta_{c'}$: positional frequency; $\delta_{c'}$: learnable phase shift

Table 10: Comparison of typical Relative Positional Encodings (RPEs). Content–position interaction indicates that the position-dependent contribution to the attention score depends on the input content, rather than being a content-independent additive bias. This includes fixed rotations applied to content vectors and does not imply learned or adaptive position indices. Learnable refers to parameters specific to the positional mechanism, excluding the standard query/key projections. Unless otherwise specified, bold lowercase vectors are represented as row vectors; in particular, $\mathbf{q}_i, \mathbf{k}_j \in \mathbb{R}^{1 \times d}$ denote the query and key at positions i and j . The function f_η denotes a parameterized function. For Transformer-XL, $\mathbf{r}_{i-j}$ denotes a sinusoidal relative-position vector, whereas for RoPE, $\mathbf{R}_i$ denotes the rotation matrix associated with position i . For Inkling, $\mathbf{r}_i = \mathbf{x}_i \mathbf{W}_r$ is a per-head relative feature produced by a fourth projection of the hidden state alongside the query, key, and value; it mixes a bank $\mathbf{P}$ of d_r learnable bias-vs-distance profiles (shared across heads and tokens) into one bias value per backward distance, and no positional term is applied beyond the maximal encoded distance L_r (512 for sliding-window layers and 1,024 for global layers). For RoPE, $\mathbf{q}_{i,c}$ and $\mathbf{k}_{j,c}$ denote the c -th two-dimensional components, with r_* and ϕ_* denoting their magnitudes and phases. For PoPE, $\tilde{q}_{i,c'}$ and $\tilde{k}_{j,c'}$ denote the complex-valued components. The symbols θ_c and $\theta_{c'}$ denote positional frequencies, and $\delta_{c'}$ is a learnable phase shift.

force” represented by a neural network—that governs the evolution of the position vector along the sequence. This is formulated as an integral equation: $p(t) = p(s) + \int_s^t h(\tau, p(\tau)) d\tau$. The learned dynamics provide a way to generate positional vectors beyond a finite lookup table; the ability to evaluate this encoding at arbitrary positions does not guarantee model quality at arbitrary sequence lengths.

10.2 Relative Positional Encoding

Relative Positional Encoding (RPE) models positional relationships between tokens in terms of their relative displacement rather than their absolute coordinates. The idea of encoding relative rather than absolute time has early roots in neural models for representing compressed event streams, where a higher-level network received the time elapsed since the last unpredictable event as input [409]. For self-attention, relative position representations were introduced by Shaw et al. [431], who add learned embeddings of the clipped relative distance $i - j$ to the keys and values inside the attention computation. Subsequent designs include additive relative biases, position-dependent rotations, and content-conditioned positional mechanisms. In modern Transformers, RPE is typically incorporated directly into the self-attention computation, allowing the interaction between a query at position i and a key at position j to depend explicitly on their relative position.

Compared with APE, which associates positional information with individual token representations, RPE treats position primarily as a property of token interactions. Different methods realize this principle in different ways, ranging from additive modifications of attention logits to transformations of query and key representations. Table 10 summarizes representative RPE methods, their attention formulations, adaptivity, and learnable components. We next discuss these approaches according to how positional information is incorporated into the attention mechanism.

Additive RPE. One major family of RPE methods incorporates positional information by adding position-dependent terms to the attention logits. An early representative is Transformer-XL [69], which reformulates the relative position representations of Shaw et al. [431] so that they can be reused across segments in its segment-level recurrence mechanism. Its relative attention score is written as

$$\text{logits}_{i,j} = \mathbf{q}_i \mathbf{k}_j^\top + \mathbf{q}_i (\mathbf{r}_{i-j} \mathbf{W}_{k,r})^\top + \mathbf{u} \mathbf{k}_j^\top + \mathbf{v} (\mathbf{r}_{i-j} \mathbf{W}_{k,r})^\top$$

where $\mathbf{r}_{i-j}$ is a fixed sinusoidal encoding of the relative displacement, $\mathbf{W}_{k,r}$ is a learnable projection matrix for the relative positional representation, and $\mathbf{u}$ and $\mathbf{v}$ are learnable global content and positional bias vectors. This decomposition separates content-based and position-dependent interactions and became an influential formulation for incorporating relative position directly into attention logits.

Subsequent methods simplify this idea by expressing positional effects as an additive bias,

$$\text{logits}_{i,j} = \mathbf{q}_i \mathbf{k}_j^\top + b(i, j),$$

with different choices of $b(i, j)$ corresponding to different positional inductive biases and degrees of adaptivity. T5 [381], for example, uses a learned scalar bias b_{i-j} , typically shared over buckets of relative distances. ALiBi [361] removes learned positional parameters altogether and imposes a fixed linear penalty that increases with token distance. KERPLE [54] and Sandwich [55] generalize this distance-based view through more flexible kernel-inspired bias functions.

More recent methods further increase the flexibility of the positional bias. FIRE [254] parameterizes the bias with a small learnable function f_η of normalized relative distance, allowing the bias profile to adapt rather than being fixed or discretized. DAPE [575] extends this direction by conditioning the positional bias not only on relative position but also on the current query–key interactions, thereby introducing content-adaptive positional modulation. CoPE [124] similarly moves beyond purely distance-based positioning by deriving contextual positions from attention interactions themselves. Inkling [484] instantiates this content-adaptive additive design at scale: each token emits an extra relative feature $\mathbf{r}_i = \mathbf{x}_i \mathbf{W}_r$ alongside its query, key, and value, which mixes a shared bank of learnable bias-vs-distance profiles $\mathbf{P}$ into a per-pair bias $\mathbf{r}_i \mathbf{P}_{:, i-j}$, so that the bias-vs-distance curve is dynamically composed per query token rather than fixed across the input; the bias is applied only within a maximal encoded distance L_r , beyond which no positional term is imposed.

Transform-based RPE. A second family of RPE methods incorporates positional information by transforming or reparameterizing the query and key representations before their interaction, rather than by adding a separate positional bias to the attention logits. In general, these methods make the query–key interaction position dependent through transformations associated with their respective positions.

Rotary Position Embedding (RoPE) [450] is the most prominent example of this approach and has been widely adopted in modern LLMs, including the LLaMA family [486, 487, 295, 296]. RoPE encodes position by applying position-dependent rotations to the query and key vectors. Let $\mathbf{R}_i$ and $\mathbf{R}_j$ denote the rotation matrices associated with

positions i and j , respectively. Following our row-vector convention, the transformed query and key are $\mathbf{q}_i \mathbf{R}_i^\top$ and $\mathbf{k}_j \mathbf{R}_j^\top$, and their interaction satisfies

$$(\mathbf{q}_i \mathbf{R}_i^\top)(\mathbf{k}_j \mathbf{R}_j^\top)^\top = \mathbf{q}_i \mathbf{R}_{j-i} \mathbf{k}_j^\top,$$

where $\mathbf{R}_i^\top \mathbf{R}_j = \mathbf{R}_{j-i}$ follows from the composition property of rotation matrices. Thus, although RoPE applies an absolute position-dependent transformation to each query and key individually, their interaction depends on the relative displacement $j - i$. In this way, relative positional structure is incorporated directly into the query-key interaction.

A complementary interpretation is obtained by viewing each two-dimensional RoPE component in polar coordinates. Let $\mathbf{q}_{i,c}$ and $\mathbf{k}_{j,c}$ denote the c -th two-dimensional components of the query and key, where $c = 1, \dots, d/2$. Their contribution to the attention score can be written as

$$r_{\mathbf{q}_{i,c}} r_{\mathbf{k}_{j,c}} \cos \left((j-i)\theta_c + \phi_{\mathbf{k}_{j,c}} - \phi_{\mathbf{q}_{i,c}} \right),$$

where $r_{\mathbf{q}_{i,c}}$ and $r_{\mathbf{k}_{j,c}}$ denote the corresponding magnitudes, $\phi_{\mathbf{q}_{i,c}}$ and $\phi_{\mathbf{k}_{j,c}}$ their content-dependent phases, and θ_c the positional frequency. This expression reveals that the relative positional phase $(j-i)\theta_c$ and the content-dependent phase difference $\phi_{\mathbf{k}_{j,c}} - \phi_{\mathbf{q}_{i,c}}$ jointly determine the angular component of the interaction. Content therefore affects the attention score both through the magnitudes and through the same phase in which relative position is encoded. This coupling has been characterized as content-position entanglement [127].

Polar Coordinate Positional Embeddings (PoPE) [127] modify this representation to separate the roles of content and position more explicitly. Rather than retaining the content-dependent phases of the original query and key vectors, PoPE maps each scalar component to a non-negative magnitude while constructing its phase from positional information. This yields complex-valued representations $\tilde{\mathbf{q}}_i$ and $\tilde{\mathbf{k}}_j$, with $c' = 1, \dots, d$ indexing their complex components. The resulting attention logits take the form

$$\text{logits}_{i,j}^{\text{PoPE}} = \sum_{c'=1}^d r_{\tilde{\mathbf{q}}_{i,c'}} r_{\tilde{\mathbf{k}}_{j,c'}} \cos \left((j-i)\theta_{c'} + \delta_{c'} \right),$$

where $r_{\tilde{\mathbf{q}}_{i,c'}}$ and $r_{\tilde{\mathbf{k}}_{j,c'}}$ encode content-dependent magnitudes, $\theta_{c'}$ is the positional frequency, and $\delta_{c'}$ is a learnable phase shift.

By removing the content-dependent phase from the angular interaction, PoPE assigns more distinct roles to the two factors: content primarily modulates the magnitude of each component's contribution, whereas positional information determines its phase. This yields a more explicit separation between content and position than in standard RoPE. This formulation has been shown to improve zero-shot length extrapolation over RoPE [127], although its direct complex-valued implementation introduces additional memory and bandwidth overhead; the original work further proposes an efficient implementation to mitigate this cost.

Beyond these formulations, several methods extend or reinterpret the positional transformations introduced by RoPE. XPOS [456] augments the rotary transformation with exponential scaling to improve extrapolation beyond the sequence lengths observed during training. FoPE [175] approaches positional encoding from a frequency-domain perspective, using Fourier decomposition to regulate undesirable frequency components and improve length generalization. As we will introduce in the next section, more recent work has also explored hybrid schemes that combine rotary transformations with dynamically adjusted positional representations, allowing the encoding strategy to adapt to different context lengths or input conditions.

10.3 Context Extension via Positional Adaptation

Generalizing beyond the sequence lengths observed during training remains a fundamental challenge for Transformers, with positional encoding playing an important role in determining such length generalization behavior [361, 221]. Beyond designing positional encodings with stronger intrinsic length generalization, another line of work focuses on extending the usable context range of pretrained models by adapting their positional mechanisms. These methods modify how pretrained positional representations or indices are interpreted beyond the original training range, for example through interpolation, rescaling, remapping, or restructuring of relative positions. Depending on whether additional parameter updates are required, these approaches can be broadly divided into training-based and training-free positional adaptation.

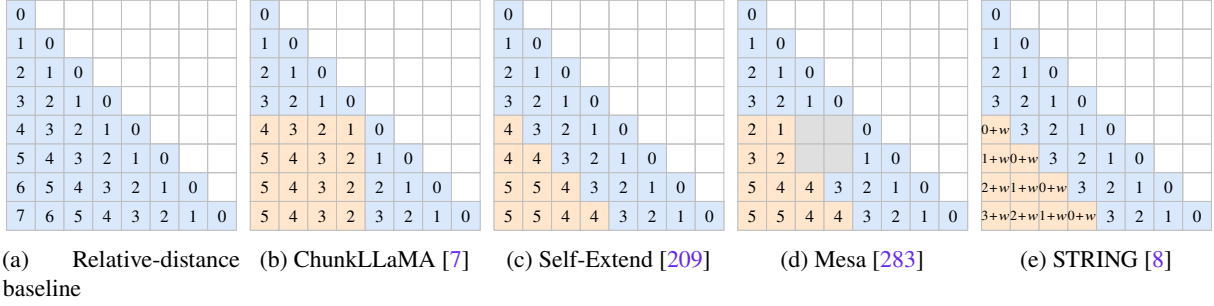


Figure 13: Comparison of training-free positional adaptation methods, shown as the relative position matrix $M[i][j]$ between query position i (row) and key position j (column) for a sequence of 8 tokens. Blue cells keep the standard relative distance $i - j$; orange cells mark positions remapped by each method so that no distance outside the pre-trained range is encountered; gray cells are excluded from attention; offset w is a hyper-parameter.

Training-Based. Positional Interpolation (PI) [48] provides a simple and effective approach to extending the context range of pretrained RoPE-based models. It linearly rescales longer positions into the pretrained positional range and then performs lightweight fine-tuning. A complementary lever is the RoPE base frequency: “NTK-aware” scaling and adjusted base frequency (ABF) [355, 524] enlarge the base (e.g., from 10^4 to 5×10^5), which lowers all rotation frequencies but affects the low-frequency dimensions the most while leaving high-frequency dimensions almost intact, and therefore extends context more gracefully than uniform interpolation for the same fine-tuning budget. Building on the general idea of adapting RoPE through positional rescaling, subsequent methods have developed more flexible scaling strategies, including YaRN [355], LongRoPE [82], and CLEX [45]. LongLoRA [51] targets the cost of the fine-tuning step rather than the positional mapping itself: it combines PI-style extension with shifted sparse attention during training and parameter-efficient LoRA updates, so that extending a 7B model to a 100K context becomes affordable on a single machine. YaRN [355] combines “NTK-by-parts” interpolation, which rescales high- and low-frequency RoPE dimensions unevenly so that the dimensions encoding local relative order are left untouched, with an attention temperature implemented directly as a scaling of the rotary embeddings; with a small amount of fine-tuning on long sequences, it extends LLaMA models to context lengths far beyond their original training length while requiring fewer fine-tuning tokens than earlier interpolation methods, and its authors report stronger perplexity and retrieval results than those methods at equal extension ratios. LongRoPE [82] further exploits non-uniform patterns across RoPE dimensions and token positions through an efficient search mechanism, which improves fine-tuning initialization and enables an $8\times$ context extension without fine-tuning in its first stage. The method further implements a progressive extension strategy, first fine-tuning for 256K context and then applying a second round of non-uniform positional interpolation search to reach 2048K without additional fine-tuning. CLEX [45] generalizes existing positional encoding scaling methods by modeling continuous dynamics through ordinary differential equations applied to the length scaling factor. This formulation overcomes the limitations of length-specific approaches and naturally extends to context lengths exceeding the training sequence length.

Training-Free. Training-free positional adaptation extends the usable context range of pretrained models by modifying positional assignments or position-dependent attention patterns at inference time, without additional parameter updates. Dual Chunk Attention (DCA, also known as ChunkLLaMA) [7] keeps the original RoPE embeddings but redesigns the relative position matrix: attention over a long sequence is decomposed into intra-chunk, inter-chunk, and successive-chunk components so that every relative distance falls within the pre-trained range, while integrating seamlessly with Flash Attention. Being orthogonal to interpolation-based methods such as PI and YaRN, DCA can be stacked with them, as done in Qwen2 [533] to reach a 128K context window. Self-Extend [209] similarly exploits positional patterns already learned by the pretrained model. It introduces bi-level attention, using neighbor attention to preserve precise relative positions for nearby tokens while grouping distant positions so that otherwise unseen relative distances are mapped to values within the pretrained range. Mesa-Extrapolation [283] processes the sequence as chunks with a triangular attention pattern: each middle chunk attends only to the first chunk and itself, with its positions re-based to directly follow the first chunk so that the encoded distances stay within the trained range; the last chunk attends to all previous tokens using Stair PE. STRING [8] shifts well-trained positions to overwrite the original ineffective positions during

inference, enhancing performance within their existing training lengths. The comparison of training-free positional adaptation methods is presented in Figure 13.

10.4 Future Directions

The journey from absolute to relative and implicit positional awareness points toward several promising research trajectories:

- **Content-Aware Positioning:** Future methods may derive more of their positional behavior from sequence content. CoPE [124] makes positional increments input-dependent, whereas DAPE [575] combines attention scores and positional biases through a learned transformation. These distinct mechanisms suggest ways to condition positional contributions on content.
- **Implicit Positional Information (NoPE):** The effectiveness of architectures that removed PE (NoPE) in Transformers challenges the necessity of explicit positional signals [188, 497]. Notably, this observation dates back to early deep autoregressive LMs, where removing PE matched or improved performance [188, 184]. These findings suggest that causal masking and successive nonlinear transformations may themselves provide sufficient structural cues for models to infer aspects of token order and relative position. A deeper understanding of when and how such implicit positional awareness emerges could inform architectures that rely less on manually specified positional inductive biases, which potentially result in more efficient and flexible models.
- **Hybrid Positional Architectures:** Explicit and implicit positional mechanisms need not be mutually exclusive. Recent architectures instead combine different positional inductive biases across the network. LLaMA 4’s iRoPE [297], for example, interleaves RoPE-equipped local-attention layers with NoPE global-attention layers, while related work explores hybrid RoPE–NoPE designs [534]. These results motivate architectures in which positional mechanisms vary across layers, heads, or attention pathways, allowing different components to specialize in local positional sensitivity or long-range interaction rather than imposing a single positional scheme uniformly throughout the model.
- **Positional Encodings for Complex Data Structures:** As Transformers are increasingly applied to non-sequential data such as graphs, 3D medical images, and multi-view scenes, the development of principled positional encodings for these complex, multi-dimensional structures remains a critical and active area of research.

Overall, the evolution of positional encoding reflects a broader shift from assigning fixed coordinates to individual tokens toward modeling position as a flexible inductive bias over token interactions. As context windows continue to expand and positional mechanisms become more adaptive, a central question is no longer simply how to encode position, but when, where, and to what extent positional information should be imposed by the architecture.

11 Tokenization

Tokenization is a fundamental process that segments text into discrete units known as tokens, which can include words, phrases, or individual symbols. This method serves as a critical first step in many linguistic and computational processes, facilitating subsequent complex modules. The granularity of these tokens can vary, ranging from bytes and characters to words and sentences. The choice of tokenization strategy significantly impacts a model’s performance, efficiency, and its ability to handle diverse languages and linguistic structures [358]. In this section, we provide an overview of tokenization approaches, including tokenizers with different granularities [299], and introduce research focused on identifying the optimal vocabulary size for effective tokenization.

11.1 Tokenizers with Different Granularities

Tokenizers differ primarily in the granularity of the units they produce; Table 11 compares the resulting trade-offs, and the following paragraphs describe each level in turn.

Character-Level Tokenizer. A character-level tokenizer splits text into individual characters, treating each as a token [422, 459, 129, 218, 562]. For example, “hello” is tokenized as [“h”, “e”, “l”, “l”, “o”]. The vocabulary is small,

typically including letters, numbers, punctuation, and spaces. Character-level tokenizers avoid the out-of-vocabulary (OOV) issue [349] for text whose characters are covered by the supported alphabet. Nevertheless, they may result in longer sequences, which increases computational complexity, and can struggle with capturing semantic meaning effectively.

Word-Level Tokenizer. Word-level tokenizers treat each word as a token [27, 301]. For example, “Hello, world!” becomes [“Hello”, “,”, “world”, “!”]. This method relies on a predefined vocabulary, with each word assigned a unique ID. It aligns well with human linguistic intuition and is effective for languages with clear word boundaries. However, it struggles with OOV words, typos, and morphological variations, often requiring a large vocabulary to account for all forms. While simple and intuitive, word-level tokenization is inefficient for morphologically rich languages and sensitive to spelling errors.

Subword-Level Tokenizer. Subword tokenizers split text into units smaller than words but larger than characters, balancing efficiency and vocabulary size [303]. For example, “unbelievable” may be tokenized into [“un”, “believ”, “able”]. Byte Pair Encoding (BPE) originated as a byte-based compression algorithm [107]. Its adaptation to subword tokenization [428] starts from characters and repeatedly merges frequent adjacent symbol pairs to construct a subword vocabulary. However, modern LLMs (e.g., GPT-2 [379]) typically utilize Byte-Level BPE (BBPE), which operates on UTF-8 bytes rather than Unicode characters. This modification ensures that any string can be processed without introducing out-of-vocabulary (OOV) tokens, as the model can fall back to individual bytes for unknown sequences. This approach allows the model to handle rare words effectively while common words are represented as single tokens. WordPiece [424, 513] iteratively merges units that maximize the likelihood of the training data when added to the vocabulary. The Unigram language model [238] offers an alternative approach to subword segmentation based on a probabilistic framework. Unlike BPE, which applies deterministic greedy merging, the Unigram model assumes that each subword occurs independently. SentencePiece [239] packages BPE and Unigram training and segmentation in a language-independent toolkit that operates on raw text without whitespace pre-tokenization, which makes it suitable for languages without explicit word delimiters and is the implementation behind many LLM tokenizers. More recently, the Over-Tokenized Transformer (OTT) [176] leaves the tokenizer unchanged but enlarges the *input* vocabulary with hashed multi-gram embeddings while keeping the output vocabulary fixed; we discuss it together with other vocabulary-scaling strategies in Sec. 11.2.

Tokenization-Free Methods. Tokenization-free methods bypass predefined subword vocabularies by processing raw text at the character or byte level. Early works laid the foundation for this paradigm: Gillick et al. [121] demonstrated multilingual language processing directly from byte sequences, and Irie et al. [186] studied byte-level CNN language models for low-resource speech recognition. A related but distinct line of *character-input* models, such as the character-aware language model of Kim et al. [225] and CharacterBERT [89], composes word-level representations from characters (e.g., via CNN-based composition) but still relies on word-level segmentation, and in Kim et al.’s case on a word-level output vocabulary. Byte-level approaches cover any byte sequence, while character-level coverage depends on the supported alphabet; both can reduce OOV issues at the tokenization stage and can improve robustness to noise, but they typically increase sequence length and computation. To make tokenization-free processing practical for Transformers, several architectures have been proposed. ByT5 [531] trains a T5-style Transformer directly on bytes (byte-to-byte), reporting competitive results with higher computational cost. CANINE [63] targets efficiency with a tokenization-free encoder that uses hashing-based embeddings and downsampling. Recent work explores hierarchical or adaptive compression: MEGABYTE [546] uses a multiscale (local/global) Transformer to model very long byte sequences; Byte Latent Transformer (BLT) [344] learns to represent text via variable-length byte patches (“latent tokens”); and MrT5 [216] applies dynamic token merging through a learned deletion gate that reduces the effective byte-sequence length.

11.2 Vocabulary Size

Current mainstream LLMs predominantly utilize subword-level tokenization methods. Historically, the vocabulary size ($|\mathcal{V}|$) has been treated as a relatively static hyperparameter, typically constrained to the range of 32K to 64K tokens (e.g., 32K for LLaMA and LLaMA 2 [486, 487]). Recent model families have moved to markedly larger vocabularies, such as 128K for LLaMA 3 [295], about 152K for the Qwen series [20, 371], and 256K for Gemma [467] (see the Vocab. column of Table 13). This design choice reflects a fundamental trade-off between *sequence efficiency* and *parameter efficiency*.

Granularity	Compression Ratio	Pros	Cons	Related Methods
Character-Level	Low (Single characters)	Small vocabulary; covers words within the supported alphabet	Long sequences, poor semantic coherence	CharCNN [562], CharRNN [218]
Word-Level	High (Full words)	Semantic units, shorter sequences	OOV issues, vocab explosion	Bengio et al. [27], RNNLM [301]
Subword-Level	Moderate (Subword units)	Balances vocabulary size and sequence length	Learned segmentation; algorithm- and corpus-dependent	BERT [80], GPT-2 [379]
Byte-Level	Low (Individual bytes)	Fixed byte alphabet; no subword OOV	Long sequences	ByT5 [531]
Adaptive Byte Patches	Variable (Grouped bytes)	Fixed byte input with compressed latent patches	Additional patching and local processing	BLT [344]

Table 11: Comparison of tokenization methods by granularity.

A larger vocabulary increases the information density per token, effectively compressing the sequence length and thereby mitigating the quadratic cost of the attention module. Conversely, increasing $|\mathcal{V}|$ linearly expands the embedding and the final output projection matrices, leading to parameter bloat and exacerbating the Softmax bottleneck [543]. This expansion also imposes significant memory bandwidth constraints and increases the computational cost of the logit generation step. While earlier scaling laws prioritized model parameters and data volume, recent studies [176, 464, 544] revisit this trade-off, underscoring the intricate role of vocabulary size in achieving a balance between performance, computational efficiency, and multilingual capabilities. These findings suggest that standard vocabulary sizes may be suboptimal for modern compute budgets and advocate for more nuanced designs, such as dynamic adaptation, the decoupling of input and output vocabularies, and compute-optimal scaling strategies.

Recent investigations [464] have shown that the compute-optimal vocabulary size grows with the compute budget following a power law, but more slowly than the number of non-vocabulary parameters: larger models benefit from expanded vocabularies, yet the vocabulary-related parameters should take up a decreasing share of the parameter budget as models scale. Here vocabulary size $V = |\mathcal{V}|$ counts token types, whereas a vocabulary matrix of hidden width d contains Vd parameters; these are distinct scaling quantities. By training models ranging from 33 million to 3 billion parameters across various vocabulary configurations, they discovered the optimal vocabulary size is intrinsically linked to the compute resources available. The findings indicate that many existing LLMs utilize suboptimal vocabulary sizes, which may restrict their performance. For example, they predicted that the optimal vocabulary size for LLaMA-2-70B [487] should be at least 216K tokens, which is seven times larger than its current vocabulary size of 32K tokens.

Scaling up vocabulary size typically improves performance but often reduces computational efficiency. To address this, the Over-Tokenized Transformer (OTT) [176] proposes a decoupling of input and output vocabularies. This approach allows for the expansion of input vocabularies to leverage multi-gram tokens while maintaining a conventional output vocabulary; the over-encoding adds hashed n -gram embeddings to each existing token position, so the token sequence and its length are unchanged. A log-linear relationship between input vocabulary size and training loss is identified, demonstrating that increasing the input vocabulary size consistently enhances model performance across the model sizes studied (OLMo2 dense models from 151M to 1B parameters and OLMoE-style MoEs). Notably, the results suggest that utilizing a large input vocabulary can achieve performance levels comparable to those obtained by doubling the model size, thus providing a computationally efficient pathway for enhancing LLM capabilities. Similarly, the SCONE method [544] extends input embedding layers by incorporating embeddings for frequently occurring n -grams, enhancing the contextual representation of input tokens. This strategy allows for the scaling of embedding layers without significantly increasing computational costs during inference. By precomputing these embeddings and storing them in off-accelerator memory, SCONE ensures efficient inference while benefiting from an expanded vocabulary.

Another emerging strategy is dynamically adjusting vocabulary size to enhance model performance. Vocabulary Curriculum Learning (VCL) [545] modifies the vocabulary during pre-training based on the model’s learning progress. By alternating between entropy-guided vocabulary expansion and model optimization, VCL enables the model to develop transferable representations across different tokenization levels. This approach fosters a hierarchical token organization, where longer tokens encapsulate predictable content, while shorter tokens focus on complex, less predictable contexts.

11.3 Future Directions

The evolution of tokenization is shifting from static, heuristic-based preprocessing toward dynamic, architecture-aware methodologies. We identify three critical trajectories for next-generation designs:

- **Compute-Optimal Vocabulary Scaling:** Compute-optimal vocabulary scaling is emerging as a necessary extension of existing scaling laws. Future research must move beyond fixed vocabulary sizes, treating the vocabulary dimension as a hyperparameter co-optimized with model depth and width.
- **Asymmetric Tokenization:** The paradigm of asymmetric tokenization offers a pathway to reconcile the trade-off between encoding context and decoding latency. By decoupling input and output vocabularies, future architectures may combine massive multi-gram input vocabularies with compact output spaces for efficient generation. Whether such input vocabularies can also shorten the sequence is a separate and open question: current over-encoding enriches the representation at each position without merging tokens, so context compression would additionally require a segmentation or merging operator that neither OTT nor SCONE provides. This asymmetry invites novel architectural adaptations, such as specialized encoder-decoder interfaces that can translate between varying granularities of semantic units.
- **Hierarchical Latent Tokenization:** The convergence of token-based and token-free methods points toward hierarchical latent tokenization. Rather than choosing between rigid subwords and raw bytes, next-generation models will likely adopt hierarchical encoders that dynamically group bytes into learned, semantic patches, akin to Vision Transformers. This approach promises to retain the universality and robustness of byte-level processing while achieving the computational efficiency of subword models, ultimately enabling truly language-agnostic foundation models.

12 Multi-Token Prediction

Mainstream LLMs are trained with the next-token prediction (NTP) objective: given a prefix, the model learns to assign a probability distribution over the single next token, and generation proceeds autoregressively, one token per forward pass. Despite its simplicity and scalability, this paradigm suffers from two fundamental limitations. First, at inference time, decoding is strictly sequential, so each generated token requires a full forward pass over the model. At small batch sizes, decoding is often memory-bandwidth-bound: the arithmetic units remain underutilized while model weights and the KV cache are repeatedly loaded from memory, so per-token latency is dictated by memory traffic rather than by arithmetic [247, 517]. Larger batches can increase arithmetic intensity and reduce the relative benefit of speculative decoding; the bottleneck also depends on context length, hardware, and implementation. Second, the NTP training signal is local and myopic, encouraging the model to latch onto local correlations and shortcut solutions rather than to plan for the future content of the sequence [123, 14]. Multi-token prediction (MTP) addresses both issues by training the model to predict several future tokens at each position. Densifying the training signal in this way encourages representations that encode longer-horizon information, and the extra prediction capability can be reused at inference time to draft multiple tokens per forward pass. To realize this acceleration, MTP is typically paired with the *draft-then-verify* paradigm of speculative decoding [446, 247]: the MTP predictions serve as cheap drafts of the next few tokens, which the model then verifies in a single parallel forward pass, amortizing the per-token memory traffic; with an appropriate acceptance rule the output distribution of the base model is preserved exactly. We classify MTP methods by how the K future predictions are factored: parallel heads that are conditionally independent given the backbone representation (Sec. 12.1), and sequential predictors that condition each step on preceding future tokens (Sec. 12.2). Joint pre-training versus retrofitting onto an existing checkpoint is an orthogonal choice of when the predictor is attached, not a third factorization. Both factorizations are consumed at inference through the same draft-then-verify procedure; methods whose main contribution is to this procedure rather than to the predictor form a separate group, inference integration (Sec. 12.3), which cuts across the two factorizations rather than adding a third.

Given a token sequence $x_{1:T}$, we use K for the total training prediction horizon, including the next token. In the parallel-head formulation, MTP predicts $x_{t+1}, \dots, x_{t+K}$ at each position t :

$$\mathcal{L}_{\text{MTP}} = - \sum_{t=1}^{T-K} \sum_{k=1}^K \log P_{\theta}(x_{t+k} \mid x_{1:t}), \quad (31)$$

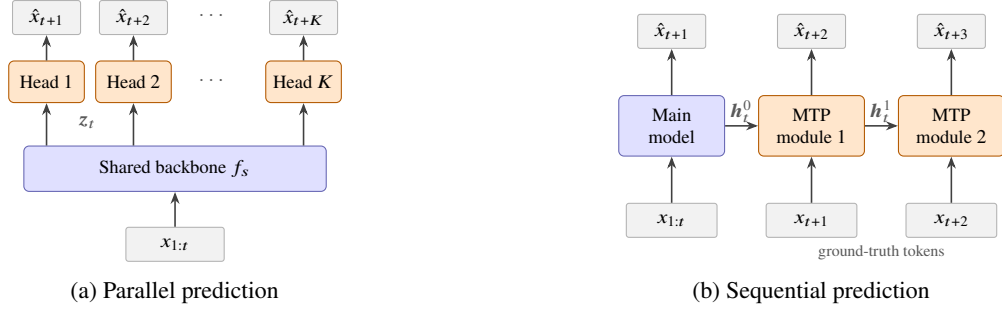


Figure 14: Comparison of the two MTP factorizations. (a) Parallel prediction attaches K heads to the same backbone representation z_t , so the K predictions are cheap but conditionally independent given z_t . (b) Sequential prediction (e.g., DeepSeek-V3 [269]) is illustrated with $D = 2$ extra modules ($K = 3$ total predictions), each combining the previous depth’s hidden state with the embedding of the intermediate token, so every prediction is conditioned on its complete preceding context; the embedding layer and the output head are shared with the main model. At training the intermediate tokens are ground truth; at inference, when the modules are retained for drafting, they are the model’s own predictions.

which reduces to NTP when $K = 1$. Sequential predictors, described below, additionally condition on intermediate tokens and therefore use a different factorization. We denote their number of extra modules by D , giving $K = D + 1$ for one module per additional position; inference rollout length is denoted separately by R and need not equal either count. The idea predates modern LLMs: blockwise parallel decoding [446] augments an autoregressive Transformer with auxiliary heads that propose the next k tokens in parallel, and ProphetNet [362] pre-trains sequence-to-sequence models with a future n -gram prediction objective via an n -stream self-attention mechanism. The central architectural question is whether the K predictions are conditionally independent given the backbone representation, or chained so that each prediction conditions on preceding future tokens. Figure 14 contrasts the two designs.

12.1 Parallel Prediction

The most direct implementation of Eq. (31) attaches K parallel output heads to a shared backbone (also called the trunk) [446, 123]. The backbone f_s produces a contextual representation $z_t = f_s(x_{1:t})$, and each head f_{h_k} , typically a single Transformer layer or feed-forward block, predicts the token k steps ahead through a shared unembedding matrix $\mathbf{W}_U \in \mathbb{R}^{d \times |V|}$ (following the row-vector convention of Sec. 4):

$$P_\theta(x_{t+k} | x_{1:t}) = \text{softmax}(f_{h_k}(z_t) \mathbf{W}_U), \quad k = 1, \dots, K. \quad (32)$$

Since all heads operate on the same backbone representation, the training cost is close to that of NTP; a memory-efficient implementation computes the forward and backward passes of the heads sequentially so that the peak memory of the large logit tensors does not scale with K [123]. The limitation of this design is that the K predictions are conditionally independent given z_t : each head predicts its target in isolation and cannot condition on the intermediate future tokens, so the heads effectively learn marginal distributions over future positions rather than the joint conditional distribution, which caps the accuracy of the predictions at larger offsets k . Follow-up work can restore communication among the heads while remaining in one parallel pass: Amphista [259] lets the future positions exchange information through bi-directional attention with staged adaptation layers, improving candidate consistency without an autoregressive rollout.

The same parallel-head structure can be attached to an already pre-trained model purely for inference acceleration. Medusa [35] fine-tunes several feed-forward decoding heads on top of a (optionally frozen) backbone, and verifies the tree of candidate continuations they propose in a single forward pass via a tree-structured attention mask. At the lightweight end of the spectrum, MuToR [115] interleaves learnable register tokens that each carry the prediction of one future position, and Samragh et al. [397] convert a pre-trained autoregressive model into a multi-token predictor by inserting mask tokens as future slots and routing the MTP behavior through gated LoRA parameters with a small sampler head, reporting speedups of about $2.5\times$ on general tasks and up to $5\times$ on code and math without quality degradation—evidence that pre-trained models already encode substantial information about future tokens. Most of these retrofit designs leave the pre-training objective of the base model untouched (Medusa-2, which jointly fine-tunes the backbone, is an exception); they trade a small amount of fine-tuning for substantial decoding speedups on existing checkpoints.

12.2 Sequential Prediction

Sequential predictors chain extra modules so that the causal chain of predictions is preserved. DeepSeek-V3 [269] instantiates this with D extra lightweight MTP modules. For $k = 1, \dots, D$, the k -th module combines the hidden state from the previous depth with the embedding of the intermediate token, and processes the result with an additional Transformer block TRM_k :

$$\mathbf{h}_t^k = \text{TRM}_k\left(\left[\text{RMSNorm}(\mathbf{h}_t^{k-1}); \text{RMSNorm}(\text{Emb}(x_{t+k}))\right] \mathbf{W}_P^k\right), \quad (33)$$

where $\mathbf{h}_t^0$ is the output of the main model, $\mathbf{W}_P^k \in \mathbb{R}^{2d \times d}$ is a fusion projection matrix, and $[\cdot; \cdot]$ denotes the concatenation of the two d -dimensional row vectors into a $2d$ -dimensional one. Each module adds a Transformer block, the fusion projection (with $2d^2$ weights), and associated normalization parameters, while sharing the embedding layer and output head with the main model. Because module k observes the token x_{t+k} before predicting x_{t+k+1} , each prediction remains conditioned on its complete preceding context, in contrast to the conditionally independent parallel heads. During training that intermediate token is the ground-truth x_{t+k} (teacher forcing). The MTP losses at all depths are averaged and added to the NTP loss with a weighting factor λ , providing a densified training signal: at inference the modules can be discarded, leaving a standard NTP model with no serving overhead, or retained and rolled out on the model’s own predictions to draft tokens for speculative decoding.

FastMTP [38] replaces the position-specific modules with a single shared-weight module applied recursively, so that the parameter count no longer grows with the prediction horizon. Since a module trained only under single-step teacher forcing degrades rapidly when unrolled on its own predictions (the acceptance rates of the first three drafted tokens fall to roughly 70%, 11%, and 2%), FastMTP applies rollout-aligned self-distillation that raises them to about 81%, 56%, and 36%, together with a language-aware vocabulary compression that reduces the drafting softmax cost, achieving an average lossless speedup of $2.03\times$ over standard decoding.

This sequential recipe has become the standard MTP component of recent open-weight LLMs. DeepSeek-V3 [269] trains with a single extra module ($D = 1, K = 2$), which improves quality on most evaluation benchmarks; when the module is retained at inference for speculative decoding, the acceptance rate of the MTP-predicted second token (x_{t+2} relative to prefix $x_{1:t}$) reaches 85%–90%, improving decoding throughput by roughly $1.8\times$. MiMo-7B [278] likewise employs an MTP module during pre-training and replicates it at inference to draft multiple tokens, reporting acceptance rates around 90%; notably, its report finds that adding more MTP layers does not consistently improve the main model’s quality but does lengthen the accepted draft, indicating that the capability-optimal and drafting-optimal prediction depths differ. Its successor MiMo-V2-Flash [279] releases three MTP layers along with the open weights, reporting an average accepted length of up to 3.6 and a decoding speedup of about $2.6\times$. Subsequent open-weight models have largely converged on this single-extra-module, DeepSeek-style recipe, though with instructive design divergences. GLM-4.5 [479] instantiates the MTP layer as an MoE layer, whereas LongCat-Flash [474] explicitly compares dense and MoE drafters and opts for a lighter dense MTP layer; it further observes that the MTP loss converges faster than the backbone objective and therefore enables it only in the middle of training, making the timing of the MTP objective itself a design variable. Qwen3-Next [477] similarly ships a native MTP module tied to speculative decoding. openPangu Ultra-MoE [463] integrates an MTP layer as well and highlights its training-system cost: the fused computation of the MTP layer, output head, and cross-entropy amounts to roughly $2.5\times$ the MoE computation and creates significant load-balancing pressure, showing that the price of native MTP is not confined to inference.

A sequential predictor can also be retrofitted onto a frozen pre-trained backbone for drafting alone. Hydra [9] keeps one head per offset but makes the k -th head condition on the candidate tokens already drafted for the preceding positions, restoring the causal chain that parallel heads omit. EAGLE [256] observes that autoregression is more predictable at the feature level than at the token level, and trains a single lightweight head that extrapolates the top-layer features of the backbone, additionally feeding in the sampled token embedding to resolve the sampling uncertainty of the feature sequence. Its successors refine both the search and the training: EAGLE-2 [257] leaves the drafter unchanged and only expands and prunes the candidate tree dynamically according to the drafter’s confidence (Sec. 12.3), whereas EAGLE-3 [258] abandons feature regression altogether, fusing hidden states from multiple layers of the backbone to predict draft tokens directly and employing a training-time test procedure that exposes the drafter to its own rollout states. HASS [558] addresses the same train–inference mismatch of recursive drafting through harmonized objective distillation and context alignment. Self-drafting can also reuse the backbone itself: Draft & Verify [557] skips a selected subset of intermediate layers of an unmodified model to produce drafts, requiring no training at all, whereas

LayerSkip [93] trains the model with early-exit losses and layer dropout so that its first layers act as the drafter and the full model as the verifier (see also Sec. 13).

12.3 Inference Integration

Regardless of the predictor structure, the multi-token predictions are usually consumed through the *draft-then-verify* paradigm of speculative decoding [446, 247, 44]: the extra heads or modules cheaply draft several future tokens, and the main model verifies them in one parallel forward pass, accepting a prefix of the draft according to an acceptance rule. The methods grouped here leave the factorization of the drafter untouched and instead improve one step of this procedure: the acceptance rule, the organization of candidates for verification, the draft length, or the alignment between drafter and target model. Most of them were introduced with a separate draft model, but the same mechanisms apply when the drafter is an MTP head or module. Speculative decoding [247] and the concurrent speculative sampling [44] define the acceptance rule: a drafted token $\hat{x}$ is accepted with probability $\min(1, p(\hat{x})/q(\hat{x}))$, where p and q are the target and draft distributions at that position, and the token at the first rejected position is resampled from the residual distribution proportional to $\max(0, p - q)$. Under this rule (or exact matching under greedy decoding), the output distribution is provably identical to standard autoregressive decoding, so MTP-based acceleration is lossless in this setting; relaxed rules such as Medusa’s typical acceptance [35] trade this guarantee for higher acceptance rates, and several of the LLM releases surveyed above do not specify which rule they use. SpecInfer [298] organizes multiple candidates per position into a token tree and verifies the whole tree in a single forward pass through tree-structured attention, and EAGLE-2 [257] keeps the EAGLE drafter unchanged but constructs the tree dynamically from the drafter’s confidence scores. SpecDec++ [177] adapts the draft length on the fly, while DistillSpec [581] and LK Losses [396] raise the acceptance rate by explicitly aligning the drafter with the target model. Blockwise parallel decoding [446] and Medusa [35] also shape this procedure, the former with greedy verification of parallel drafts and the latter with tree attention and typical acceptance, but because each introduces a new predictor, we classify both under parallel prediction (Sec. 12.1).

The achieved speedup depends on draft acceptance, drafting and verification costs, batch size, context length, hardware, and implementation. Compared with speculative decoding that relies on a separate, independently trained draft model, MTP-style self-drafting requires no standalone auxiliary model: the predictor is a small module that shares the backbone’s representations, embeddings, and output head, is trained jointly with (or fine-tuned on top of) the target model, and is served as part of it. This tight coupling is a large part of why it has become the preferred form in recent open-weight LLMs, although, as discussed in Sec. 12.4, the module must still be kept aligned when the backbone is subsequently post-trained. A comparison of representative MTP designs is provided in Table 12.

12.4 Future Directions

The trajectory of MTP suggests an evolution from an auxiliary trick into a first-class architectural component, and we identify three promising directions:

- **Prediction Horizon:** The prediction horizon remains largely heuristic: several production systems use a single extra module ($D = 1$, $K = 2$), whereas research results indicate that the optimal depth depends on model scale, data modality, and vocabulary granularity [123], and the depth that best improves model capability need not coincide with the depth that best serves drafting [278]. Treating the horizon as a dimension of the scaling laws, potentially with position-dependent schedules during training and adaptive draft lengths at inference [177], is a natural next step.
- **Hybrid Predictors:** Current designs face a trade-off between the efficiency of conditionally independent parallel heads and the accuracy of sequential predictors; hybrids that share computation across depths while preserving the causal chain [38], or that draft at the feature level as in EAGLE [256], could push acceptance rates and thus decoding speedups further.
- **Coupling to the Backbone across Post-Training:** How the MTP module stays coupled to the backbone across post-training remains largely unexplored: a drafter frozen after pre-training can drift away from the SFT- or RL-tuned target distribution, motivating designs and training recipes that keep the drafter synchronized with the evolving backbone, whether through online calibration during reinforcement learning [504] or re-distillation from the updated backbone [227, 571].

Method	Predictor Structure	Predictor Count	Draft Policy	Primary Role
Blockwise Decoding [446]	Parallel feed-forward heads	K heads	Parallel block proposals	Draft-then-verify decoding
ProphetNet [362]	n -stream self-attention	2 streams	Not used for drafting here	Pre-training objective
MTP [123]	Parallel Transformer heads; shared unembedding	$K = 4$ heads	Parallel block proposals	Training signal; self-speculative decoding
Medusa [35]	Fine-tuned MLP heads	4 extra heads	Candidate tree; parallel verification	Speculative decoding
Hydra [9]	Sequentially dependent draft heads	4 extra heads	Heads condition on preceding candidates	Speculative decoding
EAGLE [256, 257]	Feature-level autoregressive head	1 head	Static tree; dynamic in EAGLE-2	Speculative decoding
EAGLE-3 [258]	Multi-layer feature fusion; direct token drafting	1 head	Dynamic candidate tree	Speculative decoding
DeepSeek-V3 [269]	Sequential MTP; shared embedding and output head	$D = 1$ module	Optional sequential drafting	Training signal; speculative decoding
MiMo [278, 279]	Sequential MTP; inference reuse	$D = 1\text{--}3$ modules	Sequential drafting with module reuse	Training signal; speculative decoding
GLM-4.5 [479]	Sequential MoE MTP layer	$D = 1$ module	Sequential drafting	Speculative decoding
LongCat-Flash [474]	Sequential dense MTP layer	$D = 1$ module	Sequential drafting	Speculative decoding
FastMTP [38]	Shared-weight recurrent MTP head	1 shared module	Recursive rollout; aligned training	Self-speculative decoding

Table 12: Representative multi-token prediction designs and configurations, grouped by factorization. The first block is parallel heads; the second is sequential predictors. Counts refer to heads, streams, or extra sequential modules as indicated, not to tree depth or accepted length. K includes the ordinary next-token prediction; D counts extra modules, with $K = D + 1$ in the sequential training setup. The inference rollout length R depends on the draft policy and can differ from the number of distinct predictors.

13 Adaptive and Recurrent Depth

Transformer decoders conventionally apply a fixed stack of L layers to every token. Variable-depth architectures instead adapt the executed transformations, either by selecting among distinct parameterized layers or by repeatedly applying shared ones. For the former, let $m_i^{(\ell)} \in \{0, 1\}$ indicate whether token i executes layer ℓ . Its effective depth is

$$L_i = \sum_{\ell=1}^L m_i^{(\ell)}. \quad (34)$$

The standard decoder has $m_i^{(\ell)} = 1$ and hence $L_i = L$. Early exit restricts execution to a prefix of the stack, layer reduction uses a shallower subnetwork, and conditional depth allows token–layer participation to vary. Depth-recurrent architectures realize additional effective depth differently, by revisiting shared transformations. Figure 15 contrasts the three families.

13.1 Depth Reduction and Early Exit

Early-exit methods terminate computation once an intermediate representation is sufficiently predictive. If token i exits at depth e_i , then $m_i^{(\ell)} = \mathbb{1}[\ell \leq e_i]$ (Figure 15a). Depth-Adaptive Transformers learn intermediate decoder predictions and dynamically select depth in sequence-to-sequence translation, whereas CALM uses calibrated confidence criteria for token-wise early exit in language-model decoding [91, 425]. A per-token exit depth does not by itself define a runnable causal decoder: if token i exits at layer e_i and a later token j proceeds to a layer $\ell > e_i$, the self-attention at layer ℓ needs keys and values of token i that were never computed. Both methods resolve this by state propagation, copying the hidden state of the exit layer to all skipped layers and applying each layer’s own key and value projections to it; CALM reports that this approximation costs little, whereas copying the already projected keys and values of the exit layer degrades quality markedly [91, 425].

Depth can also be reduced at the network level. LayerDrop trains with structured layer dropout, making a model robust to subnetworks of different depths and enabling shallower extraction at inference time [95]. LayerSkip combines layer dropout with intermediate prediction losses; its early layers draft tokens that deeper layers verify, linking depth reduction

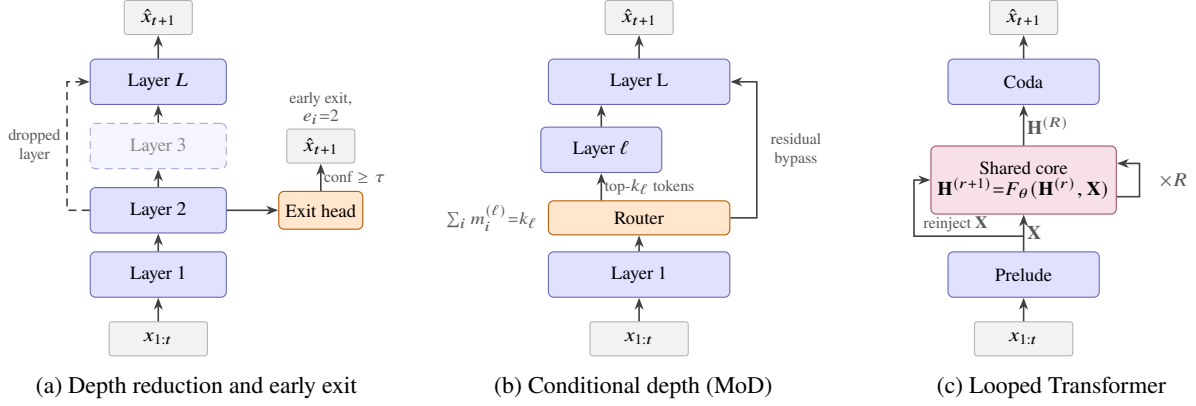


Figure 15: Three ways of varying executed depth. Blue blocks are distinct parameterized layers, orange blocks are auxiliary decision or prediction modules, and purple denotes a block whose parameters are shared across repeated applications. (a) Layer reduction (e.g., LayerDrop [95]) removes whole layers for all tokens, letting the residual stream bypass them; early exit (e.g., CALM [425]) attaches intermediate prediction heads and terminates a token once an exit criterion is met, so that $m_i^{(\ell)} = \mathbb{1}[\ell \leq e_i]$. (b) Mixture-of-Depths [384] places a router before a conditional block: an input-dependent top- k_ℓ subset of tokens executes the block while the remaining tokens take the residual bypass, keeping the active compute per layer fixed as in Eq. (35). (c) A looped Transformer (e.g., Huginn [111]) applies a shared core R times to a latent state, with the prelude representation $\mathbf{X}$ reinjected at every recurrence as in Eq. (36); the recurrence count can be fixed, assigned per token by a router [16], or determined by a convergence criterion [318].

to self-speculative decoding (Sec. 12) [93]. Thus, early exit allocates a prefix per token, while layer reduction typically selects a subnetwork shared across tokens.

13.2 Conditional Depth over Distinct Layers

Conditional depth allows $m_i^{(\ell)}$ to depend on the representation of token i at layer ℓ . Mixture-of-Depths (MoD) routes only a fixed-capacity subset of tokens through selected Transformer blocks; other tokens follow the residual bypass [384] (Figure 15b). For a sequence of T tokens and conditional-layer capacity k_ℓ ,

$$\sum_{i=1}^T m_i^{(\ell)} = k_\ell. \quad (35)$$

The active tokens are input dependent, while each conditional block retains an approximately fixed active-compute budget. The sequence-wide top- k selection used during training is non-causal, since a token’s membership depends on other tokens’ scores. For autoregressive inference, MoD instead learns a causal approximation through an auxiliary router loss or a separate predictor [384].

Conditional computation need not alter layer participation. CoLT5 applies light attention and feed-forward computation to all tokens but routes selected tokens through additional heavy branches [2]. It therefore changes computation *within* a layer, whereas MoD changes whether a token executes the principal block transformation. Routing is also independent of parameter sharing: it may select distinct layers, within-layer branches, or repeated applications of a shared block, as in Mixture-of-Recursions below. This distinction complements the expert-routing perspective in Sec. 7.

13.3 Looped (Depth-Recurrent) Transformers

A looped, or depth-recurrent, Transformer repeatedly applies a shared transformation to an evolving latent state:

$$\mathbf{H}^{(r+1)} = F_\theta(\mathbf{H}^{(r)}, \mathbf{X}), \quad r = 0, \dots, R-1. \quad (36)$$

Increasing R adds computational depth without introducing a separate parameter set at every step (Figure 15c). This recurrence acts along model depth and should not be conflated with sequence recurrence, which propagates state across positions or sequence blocks to summarize history (Sec. 6).

The Universal Transformer established this design by repeatedly applying shared Transformer computation and using Adaptive Computation Time for symbol-wise halting over recurrent depth [77, 130]. Later analyses showed that looped Transformers can implement iterative algorithms and latent multi-step reasoning [120, 535, 108, 399]. Recent language models such as Huginn and Ouro scale this principle as an adjustable axis of latent computation [111, 587].

Recurrence Topology. Architectures differ in the granularity and interfaces of the repeated computation. Huginn uses a non-recurrent prelude, a shared recurrent core, and a non-recurrent coda; the prelude representation is reinjected into the core to anchor the evolving state [111]. Such designs separate unique parameter depth from executed depth while preserving dedicated input and output transformations.

Recurrence Allocation. The recurrence count may be fixed, sampled during training, chosen globally at inference time, or assigned conditionally. Huginn trains the same core under varying recurrence counts [111]. Mixture-of-Recursions routes tokens to different recursion depths, combining parameter sharing with token-level computation allocation [16]. Fixed-Point Reasoners instead stop recurrent refinement according to a convergence criterion [318]. Conditional routing and recurrent depth are therefore compatible rather than competing categories.

Functional Differentiation under Sharing. Strict weight tying does not imply identical behavior across recurrences because the latent state changes at each step. Timestep embeddings can further condition the shared block on effective depth, while recurrence-specific low-rank modules partially relax weight tying [527, 15]. These mechanisms determine how strongly successive recurrences can specialize; this is distinct from deciding how many recurrences to execute.

13.4 Training and Scaling Variable Depth

Variable-depth models must remain predictive and stable across the depths used in training and deployment. The main issues are supervision of intermediate states, credit assignment through repeated computation, dynamical stability, and the parameter–compute trade-off.

Supervision and Credit Assignment. Intermediate objectives make shallow states directly predictive: LayerSkip supervises intermediate layers, Ouro predicts at successive recurrent steps and learns an exit distribution, and Fixed-Point Reasoners use deep supervision across truncated recurrent windows [93, 587, 318]. Full backpropagation differentiates through every recurrence but increases activation memory and extends the Jacobian chain. Huginn instead retains long forward trajectories while backpropagating only through a final recurrence window [111]. For equilibrium models, implicit differentiation avoids storing the complete solution trajectory, and phantom gradients approximate this backward computation through damped unrolling or truncated Neumann series [21, 114, 501]. Forward stopping and backward credit assignment are thus separable choices.

Dynamical Stability. Repeated transformations can amplify unstable modes or induce oscillation. Ouro reduced its training depth from eight to four recurrences after observing instability [587]. Fixed-Point Reasoners promote convergence through residual scaling, contractive initialization, and damping [318]; STARS regularizes the recurrent Jacobian’s spectral radius to encourage asymptotically stable fixed points [542]. Stability is consequently a prerequisite for using additional recurrence, not merely a numerical detail.

Depth Extrapolation. Numerical stability alone does not ensure useful predictions beyond the training-depth range. LoopFormer conditions recurrence on trajectory time and step size and enforces consistency across trajectories of different lengths, improving operation under multiple compute budgets [201]. Allocation determines the requested depth; extrapolation determines whether computation at that depth remains reliable.

Scaling Trade-offs. Parameter sharing reduces unique parameter count but not the cost of repeated attention and feed-forward computation. Ouro, for example, shares KV caches across recurrent steps to reduce decoding memory [587]. Iso-depth experiments quantify the capacity of shared recurrence through a recurrence-equivalence exponent φ : $\varphi = 0$ denotes no capacity gain and $\varphi = 1$ full equivalence to unique depth. A reported estimate of $\varphi = 0.46$, reduced to 0.38 with truncated backpropagation, indicates that recurrence provides partial rather than one-for-one replacement of independently parameterized layers [426]. These values are architecture- and training-dependent, but they expose the central exchange between parameter capacity and repeated computation.

13.5 Future Directions

Adaptive and recurrent depth turn the number of executed layers into an allocatable resource, but the mechanisms reviewed above leave several questions open:

- **Stable Training at Greater Recurrent Depth:** Current systems either cap the recurrence count during training or rely on damping, contractive initialization, and spectral regularization to keep repeated computation stable (Sec. 13.4). Whether these remedies scale to substantially deeper recurrence, and at what cost in expressiveness, remains untested at LLM scale.
- **Depth Extrapolation and Budget-Conditioned Inference:** A model should remain reliable at inference budgets not emphasized during training, so that the same weights can serve latency-critical and quality-critical requests. Trajectory-consistency objectives and time-conditioned recurrence are first steps; principled ways to choose the executed depth per token or per request are still lacking.
- **Joint Allocation across Depth, Tokens, and Experts:** Conditional depth, multi-token prediction (Sec. 12), and expert routing (Sec. 7) all decide how much computation a token receives, yet they are currently designed and tuned separately. Unifying these decisions under a single budget, and understanding their interactions with the KV cache and with batching, is a natural next step.
- **Evaluation under Matched Compute:** Recurrence-equivalence estimates such as $\varphi \approx 0.4$ show that shared recurrence is only a partial substitute for unique depth. Progress should be reported under matched training and inference compute: additional recurrence is useful only when it yields predictable gains without transferring disproportionate cost to decoding.

14 Popular Architectures

Representative large language model (LLM) families provide a system-level view of how the components surveyed above are combined in practice. We first trace the development of open-weight LLMs and the architectural precedents that shaped them, emphasizing shifts in the dominant model configuration rather than successive increases in model size. We then identify recurring design patterns that cut across individual families. Detailed configurations are reported in Table 13 for dense models and Table 14 for MoE models.

14.1 Evolution of Open-Weight LLMs and Their Precursors

Architectural Foundations and Open Replication. The development of open-weight LLMs is not a self-contained lineage: several architectural and scaling choices first became influential in models that either predate the current open-weight families or whose weights were not openly released. T5 [381], released with open weights, formulated a unified text-to-text framework with an encoder-decoder Transformer, whereas the closed GPT-3 [34] demonstrated the few-shot behavior of a decoder-only model at substantially larger scale. Jurassic-1 [262] and Gopher [380] continued this emphasis on parameter growth, while Chinchilla [168] showed that training data and model size must be allocated jointly under a fixed compute budget. PaLM [61] combined scaling with architectural choices such as parallel attention and feed-forward computation, multi-query attention, SwiGLU, and RoPE. Openly released OPT [561] and BLOOM [510] subsequently made GPT-style large-model research more accessible, with BLOOM also adopting ALiBi [361] rather than learned absolute positional embeddings. Together, these models illustrate the decoder backbone, training-scale considerations, and initial efficiency mechanisms that also appear in later open-weight families.

Convergence on a Reusable Dense Baseline. The release of the LLaMA family [486, 487, 295, 296] marked a shift from reproducing large GPT-style models to consolidating a reusable dense configuration. Its combination of a decoder-only layout, Pre-RMSNorm, SwiGLU, and RoPE became common across subsequent families, while the larger LLaMA 2 variant adopted grouped-query attention (GQA) and LLaMA 3/3.1 increased both model and training-data scale. This convergence did not eliminate variation. Pythia [31] emphasized reproducibility by releasing intermediate checkpoints; StarCoder [253] adapted the decoder to code through multi-query attention and fill-in-the-middle training; MPT [475], Falcon [5], and Baichuan 2 [532] retained distinct positional, block, or output-head choices. Mistral [205] combined GQA with sliding-window attention. Most Qwen and Qwen1.5 variants used MHA, although the later Qwen1.5-32B and 110B models adopted GQA; Qwen2 and Qwen2.5 then used GQA across their model scales [20, 476, 533, 370].

The larger dense DeepSeek LLM [30] explored a relatively deeper and narrower configuration. The resulting ecosystem shared a recognizable dense baseline while varying its attention cost, training transparency, task specialization, and deployment target.

Diversification of Capacity and State. Sparse capacity developed in parallel and later became a prominent alternative to dense scaling. Switch Transformer [96], GLaM [86], and CPM-2 [568] demonstrated token-routed MoE models at large total parameter counts, although they differed in backbone and routing configuration. OpenMoE [530] made model weights, training code, and routing analysis available, while Mixtral [312, 313] made sparse MoE available as a general-purpose open-weight model family. DeepSeekMoE [68] divided experts more finely and separated shared from routed experts; DeepSeek-V2 [73] retained this expert organization while adding Multi-head Latent Attention (MLA) to compress the inference-time KV representation. Other open efforts investigated dense-to-MoE upcycling [533, 173], sparsity in both attention and feed-forward computation [435], and fully released training artifacts [320]. Thus, the MoE line progressed from demonstrating sparse scaling to refining routing, state cost, initialization, and reproducibility.

Efficiency constraints also produced two departures from the large dense decoder. First, compact and on-device families used smaller models as testbeds for training and memory-efficient mechanisms. Successive Gemma releases added interleaved local–global attention and related stabilization choices [467, 112, 126], while MiniCPM combined scale-transfer methods with MLA, extended positional encoding, and sparse attention in later versions [173, 340, 304, 341]. MiMo [278], openPangu Embedded [47], and GPT-OSS [339] further illustrate that compact or low-active-parameter models can incorporate auxiliary prediction, controllable inference, or mixed local–global attention rather than merely shrinking a conventional backbone. Second, recurrent and hybrid models reconsidered how sequence history is represented. Griffin [72], Jamba [263], Hymba [83], and the Nemotron-H family [331, 332] combined attention with recurrent sequence operators at layer or head level, whereas Falcon Mamba [592] removed attention entirely. Later systems extended hybrid processing through Gated DeltaNet [477, 39], Kimi Delta Attention [471], and Lightning Attention [249, 306]. These designs illustrate compact recurrent state and selective access to explicit token-level history as distinct, and potentially complementary, approaches to reducing long-sequence cost.

Joint Configuration of Capacity, State, and Prediction. From late 2024 onward, leading open-weight families increasingly combined several of these directions within a single backbone. DeepSeek-V3 [269] joined fine-grained MoE routing and shared experts with MLA and an auxiliary multi-token-prediction module; DeepSeek-R1 [74] adapted the resulting backbone for reinforcement-learning-based reasoning, and DeepSeek-V3.2 [75] added sparse token selection on top of MLA. Other families combined sparse capacity with different attention, recurrent, residual, normalization, or prediction mechanisms, including Qwen3 [371], LLaMA 4 [297], Kimi K2 [469], GLM-4.5 [479], LongCat-Flash [474], Ling-1T [267], openPangu Ultra-MoE [463], Mistral Large 3 [314], and MiMo-V2-Flash [279]. The subsequent models catalogued in Tables 13 and 14 continue this diversification through sparse or hybrid attention, recurrent state, auxiliary prediction, and multimodal extensions. The historical trajectory is therefore better characterized as a shift from establishing a common dense backbone to configuring capacity, state, and prediction jointly. The next subsection examines the recurring architectural patterns produced by this shift.

14.2 Recurring Architectural Design Patterns

The historical shifts above produced several configurations that recur across otherwise distinct model families. These patterns are defined by combinations of architectural choices rather than by individual components: each addresses a system-level constraint through coordinated choices concerning parameter capacity, sequence processing, retained state, or the prediction objective. They are neither exhaustive nor mutually exclusive, and a single model may instantiate more than one pattern.

14.2.1 Canonical Dense Decoders

The canonical dense pattern combines a decoder-only Transformer with dense feed-forward blocks and a fixed layer schedule. Within this broad configuration, Pre-RMSNorm, SwiGLU, and RoPE became common ways to specify normalization, feature gating, and positional interaction without altering the uniform token-to-layer schedule. The LLaMA family [486, 487, 295, 296] established a prominent instance of this design. Across successive releases, the family changed model and data scale, vocabulary, context length, and attention grouping without replacing the basic dense computation path. Related configurations then recurred across other model families. This persistence identifies the

pattern empirically as a common reference configuration, although recurrence alone does not explain why it remained useful.

A structural explanation begins with the uniform computation path. Every token traverses the same sequence of attention and dense feed-forward blocks, so the operator schedule and parameterized modules are fixed independently of the input. At the same time, individual components can vary without reorganizing the entire backbone. LLaMA 2 used MHA in its smaller variants and GQA in the 70B model, whereas LLaMA 3 applied GQA across its released scales [487, 295]. Mistral [205] combined GQA with sliding-window attention and a rolling KV cache, bounding the explicit attention span at each layer while retaining dense feed-forward computation. The dense Qwen2 and Qwen2.5 families [533, 370] likewise used GQA across model scales. These alternatives change the number of key-value heads or the set of past positions available to attention, but not the fixed traversal of dense parameter blocks. The regular execution graph and stable interfaces among attention, feed-forward, and residual components help explain the pattern’s value for implementation and architectural comparison. Model releases do not, however, isolate the causal contribution of these properties to adoption; the pattern is therefore a persistent reference configuration rather than evidence of a universally optimal design.

14.2.2 Sparse Capacity with Compressed Attention State

Large decoders face at least two distinct scaling pressures. Increasing feed-forward capacity enlarges the parameter set and the cost of applying it, whereas long-context decoding increases the attention state stored and accessed for each generated token. Sparse MoE and efficient attention act on these pressures at different locations. MoE routing selects a small subset of experts for each token, decoupling total parameter capacity from the parameters activated on that token. KV compression, local attention, and sparse token selection instead change the representation or retrieval of sequence history. A model can therefore increase total capacity without proportionally increasing per-token feed-forward computation, yet this alone does not reduce its attention cache; conversely, compressing the cache does not make a dense feed-forward block sparse.

DeepSeek-V2 [73] is a clear joint instance. It combines the fine-grained routed experts and shared-expert isolation developed in DeepSeekMoE [68] with Multi-head Latent Attention (MLA), which stores a lower-dimensional latent representation rather than separate full key-value states for every attention head. DeepSeek-V3 [269] retains the MoE-MLA pairing, while DeepSeek-V3.2 [75] additionally restricts which past states are selected by attention. Kimi K2 [469], openPangu Ultra-MoE [463], Mistral Large 3 [314], GLM-4.7-Flash [549], and GLM-5 [122] instantiate related MoE-MLA configurations, whereas GLM-4.5 [479] pairs its fine-grained MoE with GQA (with partial RoPE and QK-Norm) rather than latent KV compression. MiMo-V2-Flash [279] illustrates a different pairing by combining sparse feed-forward capacity with alternating local and global attention rather than latent KV compression.

These combinations should not be read as a single indivisible recipe. Expert routing introduces load-balancing, communication, and batching constraints, whereas compressed or selective attention changes memory traffic, retrieval resolution, and kernel requirements. Their recurrence instead reflects a systems-level decomposition: active parameters and retained history are separate resource budgets that often become binding in the same large model. Configuring them jointly can redistribute training and serving costs, but the co-occurrence of two mechanisms does not by itself establish that either improves the modeling quality of the other.

14.2.3 Hybrid Sequence and State Processing

Hybrid sequence architectures have been pursued for two related but distinct reasons. The first is computational: replacing most softmax-attention layers with recurrent or linear operators can reduce context-dependent state and decoding cost. The second is functional: explicit attention and recurrent state provide different forms of access to history, motivating designs that retain both rather than treating either operator as a universal replacement. On a controlled set of in-context learning tasks, MambaFormer [347] reports complementary failure modes for Mamba and Transformer models and finds that their combination improves tasks on which either model alone struggles. This result provides task-specific evidence for complementarity, not a general ordering of the two operator classes. More broadly, the computational and functional motivations often coincide, but an efficiency-oriented hybrid does not by itself establish a modeling benefit.

Attention retains past token states in an explicitly addressable memory and selects among them at read time. Recurrent, state-space, and linear-attention operators instead update a compact state as the sequence is processed, moving part of

the selection or compression decision to state construction. Hybrid design therefore concerns where explicit retrieval is retained, where history is compressed, and how the two representations exchange information. A model may resolve these choices across layers, within a layer, or by converting a pretrained attention backbone after pre-training.

Layer-wise Composition. One topology assigns the operators to different layers. Griffin [72] mixes RG-LRU recurrent blocks with local attention, so recurrent updates carry information beyond the local attention window while selected layers retain explicit short-range retrieval. Jamba [263, 468] interleaves Transformer and Mamba layers and independently places MoE layers within the stack. Its design separates three choices that are sometimes conflated: the ratio of attention to recurrent layers, the placement of sparse feed-forward capacity, and the resulting memory-throughput profile. Nemotron-H and Nemotron 3 [331, 332, 330], Qwen3-Next [477, 39], and Kimi Linear [471] adopt related layer-wise compositions using Mamba-2, Gated DeltaNet, or Kimi Delta Attention together with a smaller number of attention layers. These systems share a layer-wise topology, but their recurrent updates and attention mechanisms are not interchangeable.

Within-layer Composition. A second topology combines operators within the same layer. Hymba [83] executes attention and state-space heads in parallel, allowing each layer to form both a high-resolution attention output and a compact recurrent summary before merging them. It further combines this hybrid-head arrangement with partial sliding-window attention and cross-layer KV sharing, showing that operator topology and cache compression can be configured separately. Layer-wise interleaving and within-layer parallelism impose different interfaces: the former passes representations sequentially between operator types, whereas the latter requires their outputs to be compatible at every hybrid layer. Thus, both are hybrids, but they place the interaction between explicit retrieval and recurrent state at different points in the residual computation.

Conversion from Pretrained Attention. Hybridization need not begin during pre-training. LoLCATs [560] replaces pretrained softmax-attention modules with linear attention through attention transfer, which trains the replacement to match the original attention outputs, and then uses low-rank adaptation to compensate for the remaining approximation error. This conversion route treats hybrid or linearized processing as a modification of an existing dense checkpoint rather than as a backbone trained from scratch. It lowers the training barrier to changing the sequence operator, but it also introduces a distinct objective: preserving the behavior of the original attention module under an approximate replacement.

Resource and Systems Boundary. The resource consequence depends on how much softmax attention remains. Recurrent or linear layers can update fixed-size state without adding a new key-value pair at every time step, whereas retained attention layers still require explicit history according to their attention policy. A hybrid therefore reduces, but does not generally eliminate, context-dependent cache growth. Its realized efficiency also depends on numerical behavior, memory access, kernels, prefix caching, and compatibility with decoding procedures. The MiniMax line makes this engineering boundary visible: MiniMax-01 [249, 306] combined Lightning Attention with periodic softmax attention, whereas MiniMax-M2 [308, 453] returned to full softmax attention and attributed the decision partly to evaluation uncertainty and the less mature infrastructure for efficient attention, and MiniMax-M3 [311] subsequently moved to block-sparse MSA attention (Sec. 5.1), showing that such decisions are revisited as evaluation and infrastructure mature. Hybrid sequence processing is therefore a family of trade-offs over retrieval, compression, and implementation maturity rather than a uniform replacement for attention.

14.2.4 Prediction-Augmented Backbones

Prediction-augmented backbones modify the training signal while retaining autoregressive generation as the primary modeling framework. Standard next-token prediction supervises each representation through the immediately following token. Multi-token prediction (MTP) adds objectives or modules for additional future positions, exposing the backbone to a longer prediction horizon during training. This axis is largely orthogonal to whether the backbone is dense or sparse and to how it represents history: an auxiliary predictor can be attached after representations produced by full attention, compressed attention, or a hybrid sequence stack.

DeepSeek-V3 combines an MTP module with its MoE-MLA backbone [269]; MiMo-7B applies an additional MTP objective to a compact dense decoder [278]; and LongCat-Flash and MiMo-V2-Flash attach MTP modules to sparse backbones with different attention configurations [474, 279]. These examples establish compatibility across backbone

types, but they do not imply that MTP serves the same function as the accompanying attention or routing mechanism. Sparse capacity changes which parameters process a token, efficient attention changes which historical state is retained or retrieved, and MTP changes which future targets supervise the representation.

This distinction also separates architectural prediction augmentation from later stages of model development. Reasoning-oriented post-training changes the behavior optimized during supervised fine-tuning or reinforcement learning; switchable thinking modes control how a trained model allocates generation-time reasoning; and speculative decoding uses a proposal-and-verification procedure to accelerate inference. MTP outputs can sometimes supply tentative future tokens or features to such a procedure, but an auxiliary prediction head does not by itself define how outputs are verified or committed. Prediction augmentation is therefore best understood as a reusable backbone extension whose inference-time role depends on the surrounding training and decoding system.

The four patterns overlap increasingly in recent systems. A single backbone may combine sparse parameter activation, compressed or recurrent state, heterogeneous sequence operators, and auxiliary prediction while retaining the normalization, activation, and residual conventions of the canonical dense decoder. The model comparisons in Tables 13 and 14 document these concrete configurations. Their overlap also motivates the unified view developed in the following section: modern architectures are better understood through the computational assumptions that their components jointly preserve or modify than through a list of independently selected modules.

Table 13: Architectural comparison of renowned dense LLMs. *Time Mixing* denotes the attention variant: MHA (Multi-head Attention), MQA (Multi-query Attention), GQA (Grouped-query Attention), and SWA (Sliding Window Attention); hybrid models that mix attention with linear recurrent layers (e.g., RG-LRU or Mamba) list both mechanisms. *Channel Mixing* denotes the feed-forward variant. *PE* denotes the positional encoding scheme. d_{ff} is the intermediate size of the feed-forward sublayer. *Vocab.* denotes the vocabulary size, taken from the embedding size of the released checkpoints where available and otherwise from the original paper; minor padding variations across scales within a series are ignored. Models of the same series are placed in adjacent rows. The *Release* column gives the initial public release date; rows covering multiple versions show the date of the earliest release. *Layers* and *Size* exclude Multi-Token Prediction (MTP) modules; the *MTP* column gives the number of MTP modules attached during training (“–” if none). For MoE variants of these models, see Table 14.

Model	Size	Layers	Emb. Size	Vocab.	d_{ff}	Time Mixing	Channel Mixing	Activation	PE	Normalization	MTP	Release	Remark
T5 [381]	60M	6	512		2048	MHA	FFN	ReLU/GEGLU	T5 Bias	Pre-RMSNorm	–	Oct 2019	Encoder–decoder
	220M	12	768		3072								
	770M	24	1024	32128	4096								
	3B	24	1024		16384								
	11B	24	1024		65536								
GPT-3 [34]	125M	12	768		3072	MHA/SWA	FFN	GELU	Learned APE	Pre-LayerNorm	–	May 2020	Alternates dense and banded sparse attention
	350M	24	1024		4096								
	760M	24	1536		6144								
	1.3B	24	2048	50257	8192								
	2.7B	32	2560		10240								
	6.7B	32	4096		16384								
	13B	40	5120		20480								
PanGu- α [552]	175B	96	12288		49152								
	2.6B	32	2560		10240	MHA	FFN	GELU	Learned APE	Pre-LayerNorm	–	Apr 2021	Extra query layer on top for prediction
	13.1B	40	5120	40000	20480								
CPM-2 [568]	207B	64	16384		65536	MHA	FFN	GEGLU	T5 Bias	Pre-RMSNorm	–	Jun 2021	Encoder–decoder
	11B	48	4096	26240	10240								
Jurassic-1 [262]	7.5B	32	4096		16384	MHA	FFN	GELU	Learned APE	Pre-LayerNorm	–	Aug 2021	Shallower-and-wider than GPT-3
	178B	76	13824	256000	55296								
Gopher [380]	44M	8	512		2048	MHA	FFN	GELU	Transformer-XL	Pre-RMSNorm	–	Dec 2021	
	117M	12	768		3072								
	417M	12	1536	32000	6144								
	1.4B	24	2048		8192								
	7.1B	32	4096		16384								
	280B	80	16384		65536								
GLaM [86]	1.7B	24	2048		8192	MHA	FFN	GEGLU	Transformer-XL	Post LayerNorm	–	Dec 2021	Dense baselines of the GLaM MoE study
	8.7B	32	4096	256000	16384								
	137B	64	8192		65536								
LaMDA [485]	137B	64	8192	32000	65536	MHA	FFN	GEGLU	T5 Bias	Post-LayerNorm	–	Jan 2022	Dialogue-specialized; tool-augmented grounding
Chinchilla [168]	70B	80	8192	32000	32768	MHA	FFN	GELU	Transformer-XL	Pre-RMSNorm	–	Mar 2022	
PaLM [61]	8B	32	4096		16384	MQA	FFN	SwiGLU	RoPE	Pre-LayerNorm	–	Apr 2022	Parallel attention+FFN block
	62B	64	8192	256000	32768								
	540B	118	18432		73728								

continued on next page

Table 13 continued from previous page

Model	Size	Layers	Emb. Size	Vocab.	d_{ff}	Time Mixing	Channel Mixing	Activation	PE	Normalization	MTP	Release	Remark
OPT [561]	125M	12	768		3072	MHA	FFN	ReLU	Learned APE	Pre-LayerNorm	–	May 2022	Open replication of GPT-3
	350M	24	1024		4096								
	1.3B	24	2048		8192								
	2.7B	32	2560		10240								
	6.7B	32	4096	50272	16384								
	13B	40	5120		20480								
	30B	48	7168		28672								
	66B	64	9216		36864								
	175B	96	12288		49152								
BLOOM [510]	176B	70	14336	250680	57344	MHA	FFN	GELU	ALiBi	Pre-LayerNorm	–	Jul 2022	LayerNorm after embedding
GLM [551]	130B	70	12288	150528	32768	MHA	FFN	GEGLU	RoPE	Post DeepNorm	–	Aug 2022	Blank-infilling objective; bidirectional prefix attention
GLM-4 [478]	9B	40	4096	151552	13696	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Jun 2024 Apr 2025	Partial RoPE; bias only in QKV
	32B	61	6144		23040								
GAL [466]	125M	12	768		3072	MHA	FFN	GELU	Learned APE	Pre-LayerNorm	–	Nov 2022	
	1.3B	24	2048		8192								
	6.7B	32	4096	50000	16384								
	30B	48	7168		28672								
	120B	96	10240		40960								
Pythia [31]	70M	6	512		2048	MHA	FFN	GELU	RoPE	Pre-LayerNorm	–	Apr 2023	Parallel attention+FFN block
	160M	12	768		3072								
	410M	24	1024		4096								
	1B	16	2048	50304	8192								
	1.4B	24	2048		8192								
	2.8B	32	2560		10240								
	6.9B	32	4096		16384								
	12B	36	5120		20480								
StarCoder [253]	15.5B	40	6144	49152	24576	MQA	FFN	GELU	Learned APE	Pre-LayerNorm	–	May 2023	Code LLM; fill-in-the-middle objective
MPT [475]	7B	32	4096	50432	16384	MHA	FFN	GELU	ALiBi	Pre-LayerNorm	–	May 2023	
Falcon [5]	7B	32	4544		18176	MQA	FFN	GELU	RoPE	Pre-LayerNorm	–	May 2023	Parallel attention+FFN block
	40B	60	8192	65024	32768	GQA						May 2023	
	180B	80	14848		59392	GQA						Sep 2023	
Falcon Mamba [592]	7B	64	4096	65024	8192	Mamba	Mamba	SiLU/SwiGLU	NoPE	Pre-RMSNorm	–	Aug 2024	Pure SSM; no attention layers
LLaMA-2 [487]	7B	32	4096		11008	MHA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Jul 2023	
	13B	40	5120	32000	13824	MHA							
	70B	80	8192		28672	GQA							
LLaMA-3 [295]	8B	32	4096	128256	14336	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Apr 2024	
	70B	80	8192		28672								
LLaMA-3.1 [296]	8B	32	4096		14336	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Jul 2024	
	70B	80	8192	128256	28672								
	405B	126	16384		53248								
Qwen [20]	1.8B	24	2048		5504	MHA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Aug 2023	Dynamic NTK + LogN attention scaling
	7B	32	4096	151936	11008								
	14B	40	5120		13696								
	72B	80	8192		24576								

continued on next page

Table 13 continued from previous page

Model	Size	Layers	Emb. Size	Vocab.	d_{ff}	Time Mixing	Channel Mixing	Activation	PE	Normalization	MTP	Release	Remark
Qwen1.5 [476]	0.5B	24	1024		2816								
	1.8B	24	2048		5504								
	4B	40	2560	151936	6912	MHA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Feb 2024	
	7B	32	4096		11008								
	14B	40	5120		13696								
	72B	80	8192		24576								
Qwen2 [533]	0.5B	24	896		4864								
	1.5B	28	1536	151936	8960	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Jun 2024	
	7B	28	3584		18944								
	72B	80	8192		29568								
Qwen2.5 [370]	0.5B	24	896		4864								
	1.5B	28	1536		8960								
	3B	36	2048		11008								
	7B	28	3584	151936	18944	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Sep 2024	
	14B	48	5120		13824								
	32B	64	5120		27648								
	72B	80	8192		29568								
Qwen3 [371]	0.6B	28	1024		3072								
	1.7B	28	2048		6144								
	4B	36	2560	151936	9728	GQA	FFN	SwiGLU	RoPE	QK/Pre-RMSNorm	–	Apr 2025	Unified thinking/non-thinking modes
	8B	36	4096		12288								
	14B	40	5120		17408								
	32B	64	5120		25600								
Qwen3.5/3.6 [372, 373]	0.8B	24	1024		3584								
	2B	24	2048		6144								
	4B	32	2560	248320	9216	Gated DeltaNet/ Gated Attention	FFN	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Feb 2026	Covers 3.5/3.6; DeltaNet:attn 3:1; partial RoPE; multimodal
	9B	32	4096		12288								
	27B	64	5120		17408								
Qwen3.8 [376]	27B	64	5120	248320	17408	Gated DeltaNet/ Gated Attention	FFN	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Aug 2026	Same backbone as Qwen3.5-27B; DeltaNet:attn 3:1; multi-step MTP; multimodal
Baichuan 2 [532]	7B 13B	32 40	4096 5120	125696	11008 13696	MHA	FFN	SwiGLU	RoPE ALiBi	Pre-RMSNorm	–	Sep 2023	Normalized output head (NormHead)
Mistral [205]	7B	32	4096	32000	14336	GQA+SWA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Sep 2023	SWA window 4096; rolling KV cache
DeepSeek LLM [30]	7B 67B	30 95	4096 8192	102400	11008 22016	MHA GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Nov 2023	67B favors depth over width
MiniCPM [173]	1.2B	52	1536	73440	3840	GQA						Apr 2024	
	2.4B	40	2304	122753	5760	MHA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Feb 2024	
MiniCPM3 [340]	4B	62	2560	73448	6400	MLA	FFN	SwiGLU	LongRoPE	Pre-RMSNorm	–	Sep 2024	
MiniCPM4 [304]	0.5B 8B	24 32	1024 4096	73448	4096 16384	GQA GQA+InfLLM	FFN	SwiGLU	LongRoPE	Pre-RMSNorm	–	Jun 2025	Covers 4/4.1; 8B adds InfLLM v2 sparse attn
MiniCPM5 [341, 342]	1B	24	1536	130560	4608							May 2026	On-device; think/no-think modes;
	2B	42	2048		6144	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Sep 2026	SFT→RL→on-policy distillation

continued on next page

Table 13 continued from previous page

Model	Size	Layers	Emb. Size	Vocab.	d_{ff}	Time Mixing	Channel Mixing	Activation	PE	Normalization	MTP	Release	Remark
Gemma [467]	2B 7B	18 28	2048 3072	256000	16384 24576	MQA MHA	FFN	GEGLU	RoPE	Pre-RMSNorm	–	Feb 2024	Tied embeddings
Gemma 2 [112]	2B 9B 27B	26 42 46	2304 3584 4608	256000	9216 14336 36864	GQA+SWA	FFN	GEGLU	RoPE	Pre+Post RMSNorm	–	Jun 2024	SWA:global 1:1; logit soft-capping
Gemma 3 [126]	1B 4B 12B 27B	26 34 48 62	1152 2560 3840 5376	262208	6912 10240 15360 21504	MQA+SWA GQA+SWA GQA+SWA GQA+SWA	FFN	GEGLU	RoPE	QK/Pre+Post RMSNorm	–	Mar 2025	SWA:global 5:1
Gemma 4 [113]	12B 31B	48 60	3840 5376	262144	15360 21504	GQA+SWA	FFN	GEGLU	RoPE	QK/Pre+Post RMSNorm	–	Mar 2026	SWA:global 5:1
Griffin [72]	14B	40	5120	–	15360	MQA/Recurrent block	FFN	GEGLU	RoPE	Pre-RMSNorm	–	Feb 2024	RG-LRU:local attn 2:1
LoLCATs [560]	8B 70B 405B	32 80 126	4096 8192 16384	128256	14336 28672 53248	GQA/Linear	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Oct 2024	LLaMA 3 linearized via attention transfer + LoRA
Hymba [83]	1.5B	32	1600	32001	5504	Mamba/SWA/GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	Nov 2024	Parallel Mamba and attn heads per layer; meta tokens
Nemotron-H [331]	8B 47B 56B	52 98 118	4096 8192 8192	131072	21504 30720 32768	Mamba-2/GQA	FFN	SqReLU	NoPE	Pre-RMSNorm	–	Apr 2025	Mamba-2/FFN/attn 24:24:4 (8B); 47B distilled from 56B
Nemotron Nano 2 [332]	9B 12B	56 62	4480 5120	131072	15680 20480	Mamba-2/GQA	FFN	SqReLU	NoPE	Pre-RMSNorm	–	Aug 2025	Mamba-2/FFN/attn 28:28:6; 9B distilled from 12B
Nemotron 3 [333]	4B	42	3136	131072	12544	Mamba-2/GQA	FFN	SqReLU	NoPE	Pre-RMSNorm	–	Mar 2026	Mamba-2/FFN/attn 21:17:4; compressed from Nemotron Nano 2 (9B)
MiMo [278]	7B	36	4096	151680	11008	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	1	Apr 2025	
openPangu Embedded [47]	1B 7B	26 34	1536 4096	153376	6144 12800	GQA	FFN	SwiGLU	RoPE	Pre-RMSNorm	–	May 2025	Fast/slow thinking

Table 14: Architectural comparison of renowned MoE LLMs. Experts column meaning: $N \uplus M/S$ means there are N experts always activated, plus M additional experts selected from S candidates, i.e., a total of $N + M$ experts are activated. For LongCat-Flash [474], $12/(512 \oplus 256)$ means each layer contains 512 FFN experts and 256 zero-computation experts, with exactly 12 experts activated per token (selected from both types). d_{ff} is the intermediate width of a single routed expert, taken from the released configurations (e.g., `moe_intermediate_size` or `expert_ffn_hidden_size`); where a model also contains dense FFNs, in leading dense layers or as a dense path alongside the experts, their width is given in the Remark column. *Vocab.* denotes the vocabulary size, taken from the embedding size of the released checkpoints where available and otherwise from the original paper; minor padding variations across scales within a series are ignored. Models of the same series are placed in adjacent rows. The *Release* column gives the initial public release date; rows covering multiple versions show the date of the earliest release. *Layers* and *Size* exclude Multi-Token Prediction (MTP) modules; the *MTP* column gives the number of MTP modules attached during training (“–” if none).

Model	Size	Layers	Emb. Size	Vocab.	Experts	d_{ff}	Time Mixing	Activation	PE	Normalization	MTP	Release	Remark
Switch Transformer [96]	7B-A0.2B	12	768		1/128	2048							
	26B-A0.7B	24	1024	32128	1/128	2816							
	395B-A11B	24	4096		1/64	10240							
	1.6T-A1B	15	2080		1/2048	6144							
CPM-2 [568]	198B-A11B	48	4096	26240	1/32	10240	MHA	GEGLU	T5 Bias	Pre-RMSNorm	–	Jun 2021	Encoder–decoder
GLaM [86]	105B-A1.9B	24	2048		2/256	8192							
	143B-A9.8B	32	4096	256000	2/64	16384							
	1.2T-A96.6B	64	8192		2/64	32768							
OpenMoE [530]	8.7B-A2.6B	16	2048	256384	2/32	8192	MHA	SwiGLU	RoPE	Pre-LayerNorm	–	Aug 2023	UL2 objective; open routing analysis
Mixtral [312, 313]	46.7B-A12.9B	32	4096	32000	2/8	14336							
	141B-A39B	56	6144		2/8	16384							
Mistral Large 3 [314]	675B-A41B	61	7168	131072	1 $\uplus$ 4/128	4096	MLA	SwiGLU	RoPE	Pre-RMSNorm	–	Dec 2025	Adopts DeepSeek-style MLA
DeepSeekMoE [68]	16.4B-A2.8B	28	2048	102400	2 $\uplus$ 6/64	1408	MHA	SwiGLU	RoPE	Pre-RMSNorm	–	Jan 2024	Fine-grained experts + shared-expert isolation
DeepSeek-V2 [73]	236B-A21B	60	5120	102400	2 $\uplus$ 6/160	1536	MLA	SwiGLU	RoPE	Pre-RMSNorm	–	May 2024	Introduces MLA (low-rank KV compression)
DeepSeek-V3.x [269]	671B-A37B	61	7168	129280	1 $\uplus$ 8/256	2048	MLA	SwiGLU	RoPE	Pre-RMSNorm	1	Dec 2024	Covers V3/V3.1
	671B-A37B	61	7168	129280	1 $\uplus$ 8/256	2048	MLA	SwiGLU	RoPE	Pre-RMSNorm	1	Jan 2025	RL reasoning model on the V3 base
DeepSeek-V3.2 [75]	671B-A37B	61	7168	129280	1 $\uplus$ 8/256	2048	MLA+DSA	SwiGLU	RoPE	Pre-RMSNorm	1	Sep 2025	DSA sparse attention on top of MLA
DeepSeek-V4 [76]	284B-A13B	43	4096	129280	1 $\uplus$ 6/256	2048							
	1.6T-A49B	61	7168		1 $\uplus$ 6/384	3072	CSA/HCA	SwiGLU	RoPE	Pre-RMSNorm	1	Jun 2026	CSA/HCA hybrid; mHC residuals
Jamba [263]	52B-A12B	32	4096		2/16	14336							
	398B-A94B	72	8192	65536	2/16	24576	Mamba/GQA	SwiGLU	NoPE	Pre-RMSNorm	–	Mar 2024	Attn:Mamba 1:7; MoE every Aug 2024 other layer
JetMoE [435]	8.5B-A2.2B	24	2048	32000	2/8	5632	MoA	SwiGLU	RoPE	Pre-LayerNorm	–	Apr 2024	Sparse routing in both attention (MoA) and FFN
MiniCPM-MoE [173]	13.6B-A4B	40	2304	122753	2/8	5760	MHA	SwiGLU	RoPE	Pre-RMSNorm	–	Apr 2024	Upcycled from MiniCPM-2.4B
Qwen2 [533]	57B-A14B	28	3584	151936	8 $\uplus$ 8/64	2560	GQA	SwiGLU	RoPE	Pre-RMSNorm	–	Jun 2024	Upcycled from Qwen2-7B

continued on next page

Table 14 continued from previous page

Model	Size	Layers	Emb. Size	Vocab.	Experts	d_{ff}	Time Mixing	Activation	PE	Normalization	MTP	Release	Remark
Qwen3 [371]	30B-A3B 235B-A22B	48 94	2048 4096	151936	8/128 8/128	768 1536	GQA	SwiGLU SwiGLU	RoPE	QK/Pre-RMSNorm	–	Apr 2025	No shared expert
Qwen3-Next [477, 39]	80B-A3B	48	2048	151936	1 $\cup$ 10/512	512	Gated DeltaNet/ Gated Attention	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Sep 2025	DeltaNet:attn 3:1
Qwen3.5/3.6 [372, 373]	35B-A3B 122B-A10B 397B-A17B	40 48 60	2048 3072 4096	248320	1 $\cup$ 8/256 1 $\cup$ 8/256 1 $\cup$ 10/512	512 1024 1024	Gated DeltaNet/ Gated Attention	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Feb 2026	Covers 3.5/3.6; DeltaNet:attn 3:1; multimodal
Qwen3.8-Max [377, 375]	2.4T-A95B	92	8192	248320	1 $\cup$ 10/512	2048	Gated DeltaNet/ Gated Attention	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Aug 2026	Built on Qwen3.5; DeltaNet:attn 3:1; open weights text-only, hosted Max adds vision and 1M ctx
OLMoE [320]	6.9B-A1.3B	16	2048	50304	8/64	1024	MHA	SwiGLU	RoPE	QK/Pre-RMSNorm	–	Sep 2024	Fully open: weights, data, code
MiniMax-01/M1 [249, 306]	456B-A45.9B	80	6144	200064	2/32	9216	Lightning Attn GQA	SwiGLU	RoPE	Post DeepNorm	–	Jan 2025	Lightning:softmax attn 7:1; M1: RL reasoning variant
MiniMax-M2.x [308, 307, 309, 310]	230B-A10B	62	3072	200064	8/256	1536	GQA	SwiGLU	RoPE	QK/Pre-RMSNorm	3	Oct 2025	Covers M2–M2.7; back to full softmax attention
MiniMax-M3 [311, 305]	428B-A23B	60	6144	200064	1 $\cup$ 4/128	3072	GQA+MSA	SwiGLU	RoPE	QK/Pre-RMSNorm	7	Jun 2026	Multimodal; MSA block-sparse attn; clamped SwiGLU; 3 dense layers
LLaMa 4 [297]	109B-A17B 400B-A17B	48 48	5120 5120	202048	1 $\cup$ 1/16 1 $\cup$ 1/128	8192 8192	GQA	SwiGLU	iRoPE	Pre-RMSNorm	–	Apr 2025	Maverick alternates MoE and dense (d_{ff} =16384) layers
openPangu Ultra-MoE [463]	718B-A39B	61	7680	153600	1 $\cup$ 8/256	2048	MLA	SwiGLU	RoPE	Pre+Post RMSNorm	1	May 2025	Sandwich-Norm; first 3 layers dense
openPangu-R [181]	74B-A15B	50	4608	153600	2 $\cup$ 8/80	1280	GQA	SwiGLU	RoPE	K/Pre+Post RMSNorm	1	Dec 2025	Sandwich-Norm; K-Norm; first 4 layers dense
openPangu-2.0-Flash [182]	92B-A6B	46	2560	151552	1 $\cup$ 8/256	1024	MLA+DSA/SWA	SwiGLU	RoPE	Pre+Post RMSNorm	3	Jun 2026	DSA:SWA 1:2; mHC
Kimi K2.x [469, 470, 315, 316]	1.04T-A32.6B	61	7168	163840	1 $\cup$ 8/384	2048	MLA	SwiGLU	RoPE	Pre-RMSNorm	–	Jul 2025	Covers K2–K2.7-Code; V3-style backbone with halved attention heads; multimodal from K2.5
Kimi Linear [471]	48B-A3B	27	2304	163840	1 $\cup$ 8/256	1024	KDA/MLA	SwiGLU	NoPE	Pre-RMSNorm	–	Oct 2025	KDA:MLA 3:1
Kimi K3 [317]	2.8T-A104B	93	7168	163840	2 $\cup$ 16/896	3072	KDA/Gated MLA	SiTU-GLU	NoPE	Pre-RMSNorm	1	Jul 2026	KDA:Gated MLA 3:1; AttnRes; latent MoE; multimodal
GLM-4.x [479]	106B-A12B 355B-A32B	46 92	4096 5120	151552	1 $\cup$ 8/128 1 $\cup$ 8/160	1408 1536	GQA	SwiGLU	RoPE	Pre-RMSNorm QK/Pre-RMSNorm	1	Jul 2025	Air/4.5/4.6/4.7; 355B adds QK-Norm
GLM-4.7-Flash [549]	30B-A3B	47	2048	154880	1 $\cup$ 4/64	1536	MLA	SwiGLU	RoPE	Pre-RMSNorm	1	Jan 2026	Lightweight local coding/agent variant

continued on next page

Table 14 continued from previous page

Model	Size	Layers	Emb. Size	Vocab.	Experts	d_{ff}	Time Mixing	Activation	PE	Normalization	MTP	Release	Remark
GLM-5.x [122, 480, 481]	753B-A40B	78	6144	154880	1 $\biguplus$ 8/256	2048	MLA+DSA	SwiGLU	RoPE	Pre-RMSNorm	1	Feb 2026	Covers 5/5.1/5.2/5.3; first 3 layers dense; 5.2 adds IndexShare; 5.3 scales post-training on the 5.2 base
GPT-OSS [339]	21B-A3.6B 117B-A5.1B	24 36	2880 2880	201088	4/32 4/128	2880 2880	GQA GQA	SwiGLU SwiGLU	RoPE RoPE	Pre-RMSNorm Pre-RMSNorm	–	Aug 2025	Clamped SwiGLU; SWA:full 1:1; attention sinks
Grok 2.5 [516]	270B-A115B	64	8192	131072	2/8	16384	GQA	SwiGLU	RoPE	Pre-RMSNorm	–	Aug 2025	Shared dense FFN in parallel with routed experts
LongCat-Flash [474]	560B-A27B	28	6144	131072	12/(512 $\oplus$ 256)	2048	MLA	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Sep 2025	Shortcut-connected MoE; dense FFN (d_{ff} =12288) in every block
LongCat-Flash-Lite [292]	68.5B-A3B	14	3072	131072	12/(256 $\oplus$ 128)	1024	MLA	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Jan 2026	+30B N-gram embeddings; dense FFN (d_{ff} =6144) in every block
LongCat-2.0 [291]	1.6T-A48B	38	8192	163840	12/(768 $\oplus$ 128)	12288 [†]	MLA+LSA	SwiGLU	RoPE	QK/Pre-RMSNorm	3	Jun 2026	+135B N-gram embeddings; LSA; [†] dense-path width, expert width to be confirmed
Ling-1T [267]	1T-A50B	4 ^{Dense} +76 ^{MoE}	8192	157184	1 $\biguplus$ 8/256	2048	GQA	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Oct 2025	Aux-loss-free sigmoid routing; 4 leading dense layers (d_{ff} =18432)
Ring-1T [473]	1T-A50B	4 ^{Dense} +76 ^{MoE}	8192	157184	1 $\biguplus$ 8/256	2048	GQA	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Oct 2025	RL reasoning variant of Ling-1T; same dense/expert widths
MiMo-V2.x [279, 521]	309B-A15B 1.02T-A42B	48 70	4096 6144	152576	8/256 8/384	2048 2048	GQA+SWA	SwiGLU	RoPE	Pre-RMSNorm	3	Dec 2025 Apr 2026	Covers Flash/2.5/2.5-Pro; SWA:global 5:1; first layer dense
Nemotron 3 [330, 334, 335]	30B-A3B 120B-A12B 550B-A55B	52 88 108	2688 4096 8192	131072	2 $\biguplus$ 6/128 2 $\biguplus$ 22/512 2 $\biguplus$ 22/512	1856 2688 5120	Mamba-2/GQA	SqReLU	NoPE	Pre-RMSNorm	– 1 1	Dec 2025 Mar 2026 Jun 2026	Mamba-2/attn/MoE hybrid; Super/Ultra add latent MoE
Gemma 4 [113]	26B-A4B	30	2816	262144	8/128	704	GQA+SWA	GEGLU	RoPE	QK/Pre+Post RMSNorm	–	Mar 2026	Dense FFN in parallel with experts; SWA:global 5:1
Hy3 [482]	295B-A21B	80	4096	120832	1 $\biguplus$ 8/192	1536	GQA	SwiGLU	RoPE	QK/Pre-RMSNorm	1	Jul 2026	Fast/slow thinking fusion; 256K ctx
Hy4 preview [483]	770B-A49B	78	6144	120832	1 $\biguplus$ 8/256	2048	Gated MLA+DSA	Clamped SwiGLU	RoPE	Pre-RMSNorm	1	Aug 2026	First layer dense; IndexCache; iHC residuals (4 streams); attention sinks; 1M ctx
Inkling [484]	975B-A41B	66	6144	201024	2 $\biguplus$ 6/256	3072	GQA+SWA	SwiGLU	Inkling RPE	QK/Pre-RMSNorm	–	Jul 2026	SWA:global 5:1; QKV short conv; expert sinks; multimodal

15 Beyond Components: A Unified View of Architectural Design

The preceding sections organized language model architectures primarily by components and technical families, including efficient attention and KV cache management (Sec. 5), linear sequence operators (Sec. 6), mixture of experts layers (Sec. 7), activation functions (Sec. 8), normalization (Sec. 9), positional encoding (Sec. 10), tokenization (Sec. 11), multi-token prediction and speculative decoding (Sec. 12), and adaptive and recurrent depth (Sec. 13). This organization is useful for understanding individual mechanisms, but it can obscure the broader assumptions about computational structure that different mechanisms preserve, refine, or relax.

This section provides a complementary reading of these architectural families by asking which assumptions of the canonical decoder they reconsider. Tokenization and positional encoding jointly define the sequence interface through which computation is organized. Mixture-of-experts layers change which parameters are activated for a given input, while early-exit and recurrent-depth models vary how much computation is allocated to it. Activation functions, normalization, and residual design refine the transformations performed along an active computational path and how these transformations compose across depth. KV compression and linear or recurrent sequence operators offer alternative answers to how history is represented and updated. Multi-token prediction and speculative execution distinguish how far ahead a representation is trained to predict from when tentative outputs are committed. We organize these developments into four broad architectural directions—processing structure, transformation dynamics, state management, and output commitment—and into the more concrete design dimensions summarized in Table 15. These dimensions are analytically distinct but not mutually exclusive, and a single mechanism may modify several of them simultaneously.

15.1 The Canonical Computational Structure and Its Implicit Design Assumptions

A standard decoder-only Transformer (Sec. 4) processes a fixed tokenization through a predetermined stack of parameterized layers. At a given layer, every token is processed by the same dense parameter blocks, although its attention weights, activations, and hidden representations remain input dependent. During incremental generation, the model retains layer-wise token-level key and value states in a growing KV cache. Each forward pass produces a next-token distribution, after which one token is selected and incorporated into the subsequent decoding state. These choices embody assumptions not only about local operators, but also about how representation, computation, state, and generation should be organized.

At a high level, these assumptions fall into four groups. First, the canonical decoder uses a fixed processing structure: the representation unit, active parameter blocks, layer schedule, and nominal computation assigned to each token are largely fixed independently of its content or difficulty. Second, it uses a fixed transformation and composition scheme: each block applies a predetermined nonlinear transformation within a fixed normalization and residual structure through which signals and gradients propagate. Third, it uses explicit history and append-only state: historical information is retained at token-level resolution, while the decoder state normally grows by appending finalized representations. Fourth, it uses one-step prediction and immediate commitment: each hidden state is primarily trained to predict the next token, and that prediction is committed before longer continuations are constructed or compared.

These assumptions define four broad architectural questions concerning how computation is organized, how transformations are expressed and composed, how historical state is represented and updated, and how future outputs are predicted and committed. Table 15 decomposes these questions into more concrete design dimensions.

15.2 Design Directions Beyond the Canonical Decoder

Following the four broader directions of architectural change, we summarize the corresponding design dimensions and associated methods introduced in the preceding sections, while clarifying the distinctions among closely related choices.

15.2.1 Flexible and Conditional Processing

A common trend across recent decoder architectures is the relaxation of uniform processing: the same computational structure no longer needs to be applied identically across all tokens, layers, or inputs. This relaxation ranges from *structural heterogeneity*, where different components are assigned different roles by design, to *conditional processing*, where those choices depend on the input or model state. Viewed along this spectrum, flexibility emerges in four related

Architectural direction	Design dimension	Canonical assumption	Alternative design	Involved components
Flexible and conditional processing	Sequence interface	A fixed tokenization and positional scheme define the units and coordinates of computation.	Finer, multi-scale, or adaptive units; alternative positional structures.	Tokenizer, embeddings and vocabulary, positional encoding (Secs. 10 and 11)
	Operator assignment	A uniform sequence-mixing operator is repeated across layers.	Heterogeneous sequence operators across layers or heads.	Hybrid composition of attention and alternative sequence operators (Secs. 5 and 6)
	Parameter activation	Every token activates the same dense parameter blocks.	Input-dependent routing among specialized parameter modules.	Dense or shared blocks, router, and experts (Sec. 7)
	Computation allocation	Each token follows a fixed computation path and receives a predetermined amount of processing.	Reduced, conditional, or recurrent depth; redistribution of computation across tokens or generation stages.	Early exit, conditional-depth routing, recurrent refinement, drafter–verifier computation (Secs. 12 and 13)
Refined transformation dynamics	Local nonlinearity	A fixed FFN parameterization (e.g., SwiGLU) is used within each active block.	Alternative nonlinearities or gate parameterizations, such as squared ReLU or clamped SwiGLU.	Feed-forward projections, activations, gating (Sec. 8)
	Depth propagation	Layers compose through fixed residual and normalization pathways.	Alternative normalization placement, residual scaling, or gating.	Normalization, residual pathways (Sec. 9)
Selective and flexible state management	History representation	History is retained as individually accessible token-level states.	Selective, compressed, recurrent, or hybrid history.	Sequence operator, KV cache, recurrent state (Secs. 5 and 6)
	State update	State grows append-only, without revising previously stored representations.	Compression, eviction, merging, branching, or rollback.	Cache and memory management, branching or rollback (Secs. 5, 6 and 12)
Lookahead and delayed commitment	Prediction horizon	Each state is optimized to predict the next token only.	Multi-step, blockwise, or feature-level prediction.	Prediction heads, auxiliary predictors, and drafters (Sec. 12)
	Output commitment	One decision is committed per forward pass.	Drafting, parallel verification, delayed commitment, or revision.	Decoding control, candidate management, speculative state handling (Sec. 12)

Table 15: Implicit design assumptions of the canonical decoder and representative directions in which recent architectures relax or refine them. The first column summarizes broad directions of architectural change, while the design dimensions distinguish the interface over which computation is defined, the operators and parameters that are activated, the amount of computation allocated, the dynamics of transformation and state propagation, and the timing of prediction and commitment. These dimensions are not mutually exclusive, and a single mechanism may affect several of them simultaneously.

dimensions: the units and coordinates over which computation is defined, the sequence operators used to process them, the parameter modules that are activated, and the amount of computation that is allocated.

The first two dimensions determine the structure through which information is processed. At the *sequence interface*, the canonical decoder applies a fixed tokenization and positional scheme uniformly, whereas recent designs explore finer, multi-scale, or adaptive units together with alternative positional structures (Secs. 10 and 11). A similar departure from uniformity appears in *operator assignment*: instead of repeatedly applying the same class of sequence-mixing operator, hybrid architectures combine different operators across layers or heads, for example quadratic attention with linear or recurrent alternatives (Sec. 14). Together, these developments make representation and interaction structures

more heterogeneous or conditional, allowing the organization and flow of information to vary rather than being fixed uniformly by the architecture.

The other two dimensions concern conditional use of model capacity and computation. *Parameter activation* determines which parameterized modules are used, as in mixture of experts architectures that route inputs among specialized modules (Sec. 7). *Computation allocation* instead determines how much processing is performed and where it is assigned. Variable-depth methods may reduce computation through early exit or layer skipping, redistribute it across tokens through conditional depth, or increase it through recurrent refinement (Sec. 13); speculative decoding redistributes computation across drafting and verification stages (Sec. 12). These mechanisms replace a fixed computation path with one that can contract, expand, or redistribute processing according to the input or generation process.

Across these four dimensions, flexibility therefore ranges from static heterogeneity in the processing structure to dynamic, input-dependent allocation of parameters and computation. The common trade-off is that relaxing a uniform processing pattern can reduce the regularity that makes dense Transformers efficient to train and serve, introducing additional overheads in routing, batching, and irregular execution.

15.2.2 Refined Transformation Dynamics

A second direction preserves the broad computational path while refining the transformations performed within it toward greater expressivity, selectivity, and more effective propagation across depth. The *local nonlinearity* dimension concerns the nonlinear operations applied within an active block. Changes to the baseline FFN nonlinearity or gating parameterization modify feature interactions, information transmission, and local expressivity without changing the overall computational path or the number of executed blocks (Sec. 8).

The *depth propagation* dimension instead concerns how these transformations evolve and compose across the layer stack. Normalization placement, residual scaling, and gated residual pathways shape the propagation of representations and gradients, as well as how successive transformations accumulate across depth (Sec. 9). This distinguishes depth propagation from computation allocation: adaptive-depth mechanisms determine how many transformations are executed, whereas normalization and residual design govern how a given sequence of transformations is composed.

15.2.3 Selective and Flexible State Management

A third departure from the canonical decoder concerns how past computation is retained and managed. Rather than treating history as a uniformly preserved, append-only record, recent architectures increasingly make state selective in what it stores and flexible in how that stored information can evolve. This shift spans two related dimensions: *history representation*, which determines the form and resolution of retained information, and *state update*, which determines the operations permitted on that information over time.

Along the first dimension, KV-compression methods preserve explicit access to past positions while reducing the amount or redundancy of stored state (Sec. 5), whereas linear recurrent models integrate history into compact representations (Sec. 6). These approaches differ in both what is stored and when selection occurs. Full-cache attention with sparse reads selects at retrieval time; permanent KV eviction selects during storage, and KV merging constructs aggregated entries. Linear recurrent models accumulate history into a compact state as it is constructed. Hybrid architectures combine the two strategies, preserving explicit access where fine-grained retrieval is useful while maintaining compact state elsewhere [263, 72, 83].

Flexibility also extends to the lifecycle of the retained state. Conventional KV caching is effectively append-only, but more adaptive mechanisms may evict, merge, or compress retained states, or branch and discard tentative states during multi-candidate generation and verification (Sec. 12). Representation and update are therefore distinct design choices: a recurrent model may strongly compress history while evolving strictly forward, whereas speculative decoding may retain detailed KV states while creating temporary branches that are later accepted or discarded. The broader trend is thus from history as a fixed execution trace toward state as a managed computational resource whose content, resolution, and persistence can be adapted to the needs of subsequent computation.

15.2.4 Lookahead and Delayed Commitment

A further shift concerns how future outputs are represented and finalized during generation. The canonical decoder tightly couples prediction and commitment: each state predicts the immediate next token, which is then incorporated

into the sequence before subsequent predictions are made. Recent architectures loosen this coupling by allowing the model to look farther ahead and by delaying when tentative predictions become part of the persistent decoding state.

These changes operate along two distinct dimensions. The *prediction horizon* concerns how much of the future a current representation is trained or structured to predict. Multi-token and block-wise methods extend this horizon through parallel prediction heads or sequential predictors. The *output commitment* dimension instead concerns how proposed continuations are handled before they become part of the accepted sequence. Rather than committing each token immediately after prediction, speculative decoding introduces tentative continuations that can be drafted ahead, organized into multiple candidates or trees, evaluated in parallel, and then verified, rejected, or revised before commitment. The methods reviewed in Sec. 12 collectively span both dimensions.

Together, these developments reflect a broader shift away from the tightly coupled one-step predict-and-commit paradigm of autoregressive generation. By separating prediction from irreversible state advancement, they expose a larger window of future computation that can be organized and evaluated before commitment, relaxing the strictly serial execution, one full target-model pass per committed token, that has traditionally constrained autoregressive decoding; with exact verification, the autoregressive factorization of the target distribution itself is left untouched, so what changes is the execution schedule, not the dependency structure of the model.

15.3 Interdependencies and Joint Design

The four directions above are analytically distinct but not independent. Changes along one direction can alter the cost, feasibility, or usefulness of choices along another, so architectural design is better understood as a joint configuration than as the independent optimization of isolated components. The recurring configurations identified in Sec. 14.2 show that these dimensions are combined in model backbones rather than selected in isolation. The architectures cited below illustrate representative joint configurations adopted in practice; their use should not be interpreted as evidence that these configurations are optimal. The purpose here is not to catalogue those configurations again, but to examine the dependencies that make particular combinations consequential.

Operator assignment shapes history representation. Operator assignment directly changes the form in which sequence history is retained. Attention preserves explicitly addressable token-level key-value states, whereas recurrent or linear operators update a compressed state as the sequence is processed. Hybrid architectures assign these forms of history representation to different layers or heads. Jamba interleaves attention and recurrent layers [263, 468], while Hymba combines attention and state-space heads within each layer together with additional KV-sharing mechanisms [83]. These architectures thereby instantiate different mixtures of explicit and compressed history within a single backbone.

Parameter activation and history representation impose coupled resource demands. Parameter activation and history representation govern different resource budgets. Sparse MoE layers decouple total parameter capacity from the parameters activated for each token, whereas compressed or selective attention changes the state retained or accessed during decoding. DeepSeek-V2 and DeepSeek-V3 combine sparse parameter activation with MLA-based KV compression [73, 269], providing concrete instances in which these two budgets are configured within one backbone. The mechanisms remain distinct in the forward computation, but they become coupled at the systems level: expert routing changes communication and batching behavior, while history representation changes memory traffic, kernel requirements, and context-dependent state growth. Their joint selection must therefore be evaluated against a shared resource budget.

Sequence interface shapes history representation and prediction horizon. Changes to the sequence interface interact directly with both history representation and prediction horizon. Specifically, the representation units define the scale at which both memory and prediction operate. Finer units increase sequence length and state growth, while coarser or adaptive units place more information into each sequential state. Consequently, both the cost of retaining history and the effective extent of a multi-step prediction horizon are relative to the underlying unit system. Context length, history representation, and prediction horizon should therefore be interpreted and designed jointly rather than solely in terms of token counts.

Depth propagation is coupled to computation allocation and operator assignment. Computation allocation depends on transformations remaining effective under variable execution paths. Early exit is useful only if intermediate

representations become sufficiently informative before the full stack is completed, while layer skipping and conditional depth require later transformations to operate on representations produced by heterogeneous computational histories. Recurrent depth imposes a different requirement: repeated application of shared transformations must remain dynamically stable while continuing to produce meaningful refinement as effective depth increases. Operator assignment introduces a related challenge, since residual and normalization pathways must compose representations produced by sequence operators with different mixing and propagation characteristics. Depth propagation is therefore coupled to both dimensions: changing which transformations are executed, how often they are applied, or how different operators are arranged changes the conditions under which representations must remain stable and compatible across depth.

Prediction horizon and output commitment affect state update and computation allocation differently. Prediction horizon and output commitment affect state update in different ways. Training-time MTP extends the future targets used to supervise a representation, but it does not by itself create provisional decoding states. DeepSeek-V3 provides a concrete configuration in which an MTP objective coexists with MLA-based history representation and sparse parameter activation [269], making the corresponding training and resource choices jointly relevant within one backbone. Delayed output commitment at inference time, by contrast, creates provisional computation beyond the accepted sequence. Longer drafts or wider candidate trees require temporary states that may need to branch, persist selectively, or be discarded or rolled back after verification. Compression can reduce this overhead, yet aggressive compression may make verification or rollback more difficult if relevant information has been removed. Output commitment also interacts with computation allocation: tentative continuations can be produced with cheaper or shallower computation and subjected to more expensive processing only when verification is required. This asymmetry further couples computation allocation to state update because states produced under different amounts of computation must remain usable when deeper processing or verification is subsequently invoked. History representation, state update, computation allocation, prediction horizon, and output commitment therefore cannot always be optimized independently.

Architectural choices remain subject to realization constraints. The dimensions above characterize the forward architecture, but the usable design space is also shaped by training and systems mechanisms. Sparse parameter activation depends on load balancing, communication, and efficient batching; recurrent or conditional computation allocation depends on stable optimization and credit assignment across the executed path; and alternative operator assignments depend on numerical behavior, optimized kernels, caching support, and compatibility with decoding procedures. The transition from the Lightning-Attention hybrid of MiniMax-01 to the full-attention design of MiniMax-M2 provides a concrete example: MiniMax attributes the later choice partly to evaluation uncertainty and the relative immaturity of efficient-attention infrastructure [249, 453], and the subsequent adoption of block-sparse MSA attention in MiniMax-M3 [311] shows that such decisions are revisited as infrastructure matures. These considerations are not additional architectural dimensions, but they determine which joint configurations are practical at scale.

More generally, these architectural directions interact through both trade-offs and complementarities. Conditional parameter activation or computation allocation can reduce executed computation while making routing and communication more prominent; extended prediction horizons and delayed output commitment can amortize sequential decoding while introducing drafting, verification, and temporary-state overhead; and compressed or recurrent history representations can reduce memory growth while changing the cost and flexibility of state access and update. At the same time, a lower state cost can make a longer prediction horizon, broader branching, or additional refinement affordable, while more stable depth propagation can support more aggressive conditional or recurrent computation allocation. The value of an individual mechanism therefore depends not only on the cost it removes, but also on the costs it introduces and the architectural choices it enables elsewhere.

16 Concluding Remarks

This survey has examined the architectural innovations that are shaping the next generation of large language models. Starting from the modern Transformer decoder (Sec. 4), we reviewed efficient attention and KV cache management (Sec. 5), linear language models (Sec. 6), mixture of experts (Sec. 7), activation functions (Sec. 8), normalization and residual connections (Sec. 9), positional encoding (Sec. 10), tokenization (Sec. 11), multi-token prediction (Sec. 12), and adaptive and recurrent depth (Sec. 13), and then examined how representative open-weight LLM families combine these components in practice (Sec. 14). Rather than restating each section, we close by summarizing what the survey as a whole suggests about where LLM architecture is heading.

The first observation is that the canonical decoder-only Transformer, with dense feed-forward blocks, a fixed layer schedule, and the now-standard combination of Pre-RMSNorm, SwiGLU, and RoPE, remains the reference configuration against which nearly every innovation is defined. Its persistence is plausibly explained by the fact that a uniform computation path with stable interfaces among attention, feed-forward, and residual components is easy to implement, scale, and compare, although, as noted in Sec. 14, model releases do not isolate the causal contribution of these properties to adoption. Most of the developments surveyed here do not discard this backbone. Instead, each selectively relaxes one of its implicit assumptions about how representation, computation, state, and generation are organized, while retaining the rest.

Viewed through the unified perspective of Sec. 15, these relaxations cluster into four broad directions. *Flexible and conditional processing* loosens the assumption that every token is processed identically: tokenization and positional encoding redefine the units and coordinates over which computation is defined; hybrid architectures assign heterogeneous sequence operators across layers or heads; mixture of experts makes parameter activation input-dependent; and early exit, conditional depth, and looped Transformers make the amount of computation per token adjustable. *Refined transformation dynamics* keeps the computational path fixed but improves what happens along it, through gated activations that shape local nonlinearity and through normalization and residual designs that govern how transformations compose across depth. *Selective and flexible state management* replaces the append-only, token-level KV cache with history that is compressed, selected, recurrent, or hybrid, and whose lifecycle admits eviction, merging, branching, and rollback. *Lookahead and delayed commitment* decouples prediction from commitment: multi-token prediction lengthens the horizon a representation is trained to predict, and speculative verification, a decoding-time procedure that architectural components such as MTP modules and draft heads are designed to support, defers the point at which tentative outputs become part of the decoding state.

The second observation is that these directions are increasingly configured jointly rather than adopted in isolation. The recurring patterns identified in Sec. 14, namely sparse capacity paired with compressed attention state, hybrid sequence and state processing, and prediction-augmented backbones, are combinations, and their value derives from how the constituent choices interact. Operator assignment determines the form in which history is represented; sparse parameter activation and compressed history draw on distinct but coupled resource budgets; the sequence interface sets the scale at which both memory cost and prediction horizon are measured; and depth propagation must remain stable under variable execution paths and heterogeneous operators. The benefit of any single mechanism therefore depends not only on the cost it removes, but also on the costs it introduces and the design choices it enables elsewhere.

The third observation is that the usable design space is bounded by realization constraints as much as by modeling considerations. Sparse activation depends on load balancing, communication, and batching; recurrent and conditional depth depend on stable optimization and credit assignment through the executed path; and alternative sequence operators depend on numerical behavior, optimized kernels, caching support, and compatibility with decoding procedures. The decision of at least one recent hybrid family to return to full attention, even as others continued to adopt hybrid designs, before moving on to block-sparse attention in its next generation (Sec. 15), illustrates that infrastructure maturity and evaluation uncertainty can outweigh efficiency arguments at scale, and that such decisions are revisited as infrastructure matures. Architectural progress therefore advances hand in hand with training recipes and systems support.

Taken together, these observations suggest that next-generation LLM architectures will be characterized less by a single replacement for attention or for the dense feed-forward block than by increasingly adaptive allocation of computation, capacity, state, and prediction across tokens, layers, and time. Several challenges remain open. Scaling laws remain incomplete for the new axes exposed by this survey: beyond existing laws for routed models in terms of expert count and granularity [62, 236], prediction horizon, recurrent depth, and state size lack comparable treatment, particularly under matched training and inference compute. Training methods must keep conditional and recurrent computation stable as it is scaled. Evaluation protocols should expose the trade-offs among expressiveness, efficiency, and stability rather than measuring any one in isolation. Meeting these challenges will require sustained interplay between architectural design, optimization, and systems engineering. We hope this survey offers a structured map of the design space to support that effort.

References

- [1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Empirical*

- Methods in Natural Language Processing*, 2023. 5, 9, 18, 26
- [2] Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontanon, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, et al. CoLT5: Faster Long-Range Transformers with Conditional Computation. In *Empirical Methods in Natural Language Processing*, 2023. 9, 65
 - [3] A Aizerman. Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning. *Automation and Remote Control*, 1964. 33
 - [4] Yaroslav Aksenov, Nikita Balagansky, Sofia Maria Lo Cicero Vaina, Boris Shaposhnikov, Alexey Gorbatovski, and Daniil Gavrilov. Linear Transformers with Learnable Kernel Functions Are Better In-Context Models. In *Association for Computational Linguistics*, 2024. 34, 35
 - [5] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. The Falcon Series of Open Language Models. Preprint arXiv:2311.16867, 2023. 67, 73
 - [6] S-I Amari. Learning Patterns and Pattern Sequences by Self-Organizing Nets of Threshold Elements. *IEEE Transactions on Computers*, 1972. 11, 30
 - [7] Chenxin An, Fei Huang, Jun Zhang, Shansan Gong, Xipeng Qiu, Chang Zhou, and Lingpeng Kong. Training-Free Long-Context Scaling of Large Language Models. In *International Conference on Machine Learning*, 2024. 9, 56
 - [8] Chenxin An, Jun Zhang, Ming Zhong, Lei Li, Shansan Gong, Yao Luo, Jingjing Xu, and Lingpeng Kong. Why Does the Effective Context Length of LLMs Fall Short? In *International Conference on Learning Representations*, 2025. 9, 56
 - [9] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-Dependent Draft Heads for Medusa Decoding. In *Conference on Language Modeling*, 2024. 9, 62, 64
 - [10] Anthropic. System Card: Claude Fable 5 & Claude Mythos 5. System card, Anthropic, 2026. URL <https://www-cdn.anthropic.com/d00db56fa754a1b115b6dd7cb2e3c342ee809620.pdf>. Published 2026-06-09; accessed 2026-07-28. 8
 - [11] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary Evolution Recurrent Neural Networks. In *International Conference on Machine Learning*, 2016. 31
 - [12] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Ré. Simple Linear Attention Language Models Balance the Recall-Throughput Tradeoff. In *International Conference on Machine Learning*, 2024. 6, 34, 35
 - [13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer Normalization. Preprint arXiv:1607.06450, 2016. 7, 9, 15, 47, 48, 49
 - [14] Gregor Bachmann and Vaishnavh Nagarajan. The Pitfalls of Next-Token Prediction. In *International Conference on Machine Learning*, 2024. 60
 - [15] Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-Wise LoRA. In *International Conference on Learning Representations*, 2025. 66
 - [16] Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron C. Courville, and Se-Young Yun. Mixture-of-Recursions: Learning Dynamic Recursive Depths for Adaptive Token-Level Computation. In *Neural Information Processing Systems*, 2025. 7, 9, 65, 66
 - [17] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*, 2015. 11

- [18] Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1983. 10
- [19] Lalit R Bahl, Peter F Brown, Peter V De Souza, and Robert L Mercer. A Tree-Based Statistical Language Model for Natural Language Speech Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1989. 10
- [20] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen Technical Report. Preprint arXiv:2309.16609, 2023. 10, 46, 49, 58, 67, 73
- [21] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep Equilibrium Models. In *Neural Information Processing Systems*, 2019. 66
- [22] Yushi Bai, Qian Dong, Ting Jiang, Xin Lv, Zhengxiao Du, Aohan Zeng, Jie Tang, and Juanzi Li. IndexCache: Accelerating Sparse Attention via Cross-Layer Index Reuse. Preprint arXiv:2603.12201, 2026. 23
- [23] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *European Conference on Computer Vision*, 2018. 22
- [24] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended Long Short-Term Memory. In *Neural Information Processing Systems*, 2024. 6, 9, 31, 33, 37, 38
- [25] Maximilian Beck, Korbinian Pöppel, Phillip Lippe, and Sepp Hochreiter. Tiled Flash Linear Attention: More Efficient Linear RNN and xLSTM Kernels. In *Neural Information Processing Systems*, 2025. 40
- [26] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to Memorize at Test Time. In *Neural Information Processing Systems*, 2025. 38
- [27] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A Neural Probabilistic Language Model. In *Neural Information Processing Systems*, 2000. 10, 11, 29, 58, 59
- [28] Adam Berger, Stephen A Della Pietra, and Vincent J Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 1996. 10
- [29] Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Gosh. Memory Layers at Scale. In *International Conference on Machine Learning*, 2025. 44
- [30] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling Open-Source Language Models with Longtermism. Preprint arXiv:2401.02954, 2024. 10, 46, 68, 74
- [31] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O’Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. In *International Conference on Machine Learning*, 2023. 67, 73
- [32] William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing Transformer Key-Value Cache Size with Cross-Layer Attention. In *Advances in Neural Information Processing Systems*, 2024. 29
- [33] Peter F Brown, Vincent J Della Pietra, Peter V Desouza, Jennifer C Lai, and Robert L Mercer. Class-Based N-gram Models of Natural Language. *Computational Linguistics*, 1992. 10
- [34] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models Are Few-Shot Learners. In *Neural Information Processing Systems*, 2020. 12, 16, 49, 52, 67, 72
- [35] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *International Conference on Machine Learning*, 2024. 7, 9, 61, 63, 64

- [36] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A Survey on Mixture of Experts. Preprint arXiv:2407.06204, 2024. 7, 40
- [37] Weilin Cai, Juyong Jiang, Le Qin, Junwei Cui, Sunghun Kim, and Jiayi Huang. Shortcut-Connected Expert Parallelism for Accelerating Mixture of Experts. In *International Conference on Machine Learning*, 2025. 9, 41, 44
- [38] Y. Cai et al. FastMTP: Accelerating LLM Inference with Enhanced Multi-Token Prediction. Preprint arXiv:2509.18362, 2025. 9, 62, 63, 64
- [39] Ruisheng Cao, Mouxian Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, et al. Qwen3-Coder-Next Technical Report. Preprint arXiv:2603.00729, 2026. 68, 70, 77
- [40] Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL <https://arxiv.org/abs/2501.14249>. 4, 8
- [41] Center for AI Safety and Scale AI. Humanity’s Last Exam Dataset. Hugging Face Dataset Repository, <https://huggingface.co/datasets/cais/hle>, 2026. Accessed 2026-07-29. 4
- [42] Chameleon Team. Chameleon: Mixed-Modal Early-Fusion Foundation Models. Preprint arXiv:2405.09818, 2024. 49
- [43] Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. Palu: Compressing KV-Cache with Low-Rank Projection. In *International Conference on Learning Representations*, 2025. 9, 28
- [44] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating Large Language Model Decoding with Speculative Sampling. Preprint arXiv:2302.01318, 2023. 9, 63
- [45] Guanzheng Chen, Xin Li, Zaiqiao Meng, Shangsong Liang, and Lidong Bing. CLEX: Continuous Length Extrapolation for Large Language Models. In *International Conference on Learning Representations*, 2023. 9, 56
- [46] Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. SepLLM: Accelerate Large Language Models by Compressing One Segment into One Separator. In *International Conference on Machine Learning*, 2025. 5, 9, 18, 20, 21
- [47] Hanting Chen, Yasheng Zeng, Wenyuan Wang, et al. Pangu Embedded: An Efficient Dual-System LLM Reasoner with Metacognition. Preprint arXiv:2505.22375, 2025. 68, 75
- [48] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending Context Window of Large Language Models via Positional Interpolation. Preprint arXiv:2306.15595, 2023. 6, 9, 56
- [49] Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. NACL: A General and Effective KV Cache Eviction Framework for LLM at Inference Time. In *Association for Computational Linguistics*, 2024. 20, 21
- [50] Yuang Chen, Cheng Zhang, Xitong Gao, Robert D. Mullins, George A. Constantinides, and Yiren Zhao. Optimised Grouped-Query Attention Mechanism for Transformers. Preprint arXiv:2406.14963, 2024. 9, 26
- [51] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. LongLoRA: Efficient Fine-Tuning of Long-Context Large Language Models. In *International Conference on Learning Representations*, 2023. 6, 9, 56
- [52] Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Léon Bottou, Zhihao Jia, and Beidi Chen. MagicPIG: LSH Sampling for Efficient LLM Generation. In *International Conference on Learning Representations*, 2025. 20, 22

- [53] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long Short-Term Memory-Networks for Machine Reading. In *Empirical Methods in Natural Language Processing*, 2016. 11
- [54] Ta-Chung Chi, Ting-Han Fan, Peter J Ramadge, and Alexander Rudnicky. KERPLE: Kernelized Relative Positional Embedding for Length Extrapolation. In *Neural Information Processing Systems*, 2022. 9, 53, 54
- [55] Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, and Peter Ramadge. Dissecting Transformer Length Extrapolation via the Lens of Receptive Field Analysis. In *Association for Computational Linguistics*, 2023. 9, 53, 54
- [56] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. Preprint arXiv:1904.10509, 2019. 49
- [57] Sai Sena Chinnakonduru and Astarag Mohapatra. Weighted Grouped Query Attention in Transformers. Preprint arXiv:2407.10855, 2024. 9, 26
- [58] Krzysztof Marcin Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking Attention with Performers. In *International Conference on Learning Representations*, 2021. 34
- [59] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-Based Models for Speech Recognition. In *Neural Information Processing Systems*, 2015. 11
- [60] Yuhong Chou, Man Yao, Kexin Wang, Yuqi Pan, Rui-Jie Zhu, Jibin Wu, Yiran Zhong, Yu Qiao, Bo Xu, and Guoqi Li. MetaLA: Unified Optimal Linear Approximation to Softmax Attention Map. In *Neural Information Processing Systems*, 2024. 9, 37, 38
- [61] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 2023. 67, 72
- [62] Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified Scaling Laws for Routed Language Models. In *International Conference on Machine Learning*, 2022. 6, 41, 84
- [63] Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. CANINE: Pre-Training an Efficient Tokenization-Free Encoder for Language Representation. *Transactions of the Association for Computational Linguistics*, 2022. 9, 58
- [64] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *International Conference on Learning Representations*, 2016. 34, 45
- [65] Diane Coyle and Lucy Hampton. 21st Century Progress in Computing. *Telecommunications Policy*, 2024. 5
- [66] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. Approximating Two-Layer Feedforward Networks for Efficient Transformers. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023. 9, 42, 43
- [67] Róbert Csordás, Piotr Piękos, Kazuki Irie, and Jürgen Schmidhuber. SwitchHead: Accelerating Transformers with Mixture-of-Experts Attention. In *Neural Information Processing Systems*, 2024. 9, 43
- [68] Damai Dai, Chengqi Deng, Chenggang Zhao, Rx Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models. In *Association for Computational Linguistics*, 2024. 6, 9, 10, 41, 43, 68, 69, 76
- [69] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In *Association for Computational Linguistics*, 2019. 9, 53, 54

- [70] Tri Dao and Albert Gu. Transformers Are SSMS: Generalized Models and Efficient Algorithms Through Structured State Space Duality. In *International Conference on Machine Learning*, 2024. 6, 9, 33, 35, 37, 38
- [71] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language Modeling with Gated Convolutional Networks. In *International Conference on Machine Learning*, 2017. 6, 46
- [72] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models. Preprint arXiv:2402.19427, 2024. 6, 9, 31, 37, 38, 68, 70, 75, 81
- [73] DeepSeek-AI. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. Preprint arXiv:2405.04434, 2024. 5, 9, 10, 18, 27, 43, 68, 69, 76, 82
- [74] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. Preprint arXiv:2501.12948, 2025. 68, 76
- [75] DeepSeek-AI, Aixin Liu, Aoxue Mei, et al. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. Preprint arXiv:2512.02556, 2025. 20, 23, 68, 69, 76
- [76] DeepSeek-AI et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. 23, 50, 76
- [77] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. In *International Conference on Learning Representations*, 2019. 7, 9, 66
- [78] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling Vision Transformers to 22 Billion Parameters. In *International Conference on Machine Learning*, 2023. 48
- [79] Stephen A Della Pietra, Vincent J Della Pietra, Robert L Mercer, and Salim Roukos. Adaptive Language Modeling Using Minimum Discriminant Estimation. In *Speech and Natural Language*, 1992. 10
- [80] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019. 6, 12, 16, 52, 59
- [81] Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, et al. CogView: Mastering Text-to-Image Generation via Transformers. In *Neural Information Processing Systems*, 2021. 7, 9, 49, 51
- [82] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens. In *International Conference on Machine Learning*, 2024. 6, 9, 56
- [83] Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, et al. Hymba: A Hybrid-Head Architecture for Small Language Models. In *International Conference on Learning Representations*, 2025. 68, 70, 75, 81, 82
- [84] Cicero dos Santos, James Lee-Thorp, Isaac Noble, Chung-Ching Chang, and David C Uthus. Memory Augmented Language Models through Mixture of Word Experts. In *North American Chapter of the Association for Computational Linguistics*, 2024. 9, 44
- [85] Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, and Yu Cheng. MoM: Linear Sequence Modeling with Mixture-of-Memories. In *International Conference on Learning Representations*, 2026. 40
- [86] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. GLaM: Efficient Scaling of Language Models with Mixture-of-Experts. In *International Conference on Machine Learning*, 2022. 9, 43, 68, 72, 76

- [87] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark. *Neurocomputing*, 2022. 7
- [88] David Eigen, Marc’Aurelio Ranzato, and Ilya Sutskever. Learning Factored Representations in a Deep Mixture of Experts. Preprint arXiv:1312.4314, 2013. 41
- [89] Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Jun’ichi Tsujii. CharacterBERT: Reconciling ELMo and BERT for Word-Level Open-Vocabulary Representations From Characters. In *International Conference on Computational Linguistics*, 2020. 9, 58
- [90] Venmugil Elango, Nidhi Bhatia, Roger Waleffe, Rasoul Shafipour, Tomer Asida, Abhinav Khattar, Nave Assaf, Maximilian Golub, Joey Guman, Tiyyasa Mitra, et al. LatentMoE: Toward Optimal Accuracy per FLOP and Parameter in Mixture of Experts. Preprint arXiv:2601.18089, 2026. 9, 41, 44
- [91] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-Adaptive Transformer. In *International Conference on Learning Representations*, 2020. 7, 9, 64
- [92] Stefan Elfving, Eiji Uchibe, and Kenji Doya. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. *Neural Networks*, 2018. 6, 9, 46, 47
- [93] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, et al. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding. In *Association for Computational Linguistics*, 2024. 7, 9, 63, 65, 66
- [94] Jeffrey L Elman. Finding Structure in Time. *Cognitive Science*, 1990. 11, 29
- [95] Angela Fan, Edouard Grave, and Armand Joulin. Reducing Transformer Depth on Demand with Structured Dropout. In *International Conference on Learning Representations*, 2020. 7, 9, 64, 65
- [96] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 2022. 6, 8, 9, 40, 41, 42, 68, 76
- [97] Jerome A Feldman. Dynamic Connections in Neural Networks. *Biological Cybernetics*, 1982. 11, 31
- [98] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. In *Neural Information Processing Systems*, 2025. 19, 20
- [99] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Identify Critical KV Cache in LLM Inference from an Output Perturbation Perspective. In *International Conference on Machine Learning*, 2026. 20, 21
- [100] Mikel L Forcada and Ramón P Neco. Recursive Hetero-Associative Memories for Translation. In *International Work-Conference on Artificial Neural Networks*, 1997. 11, 29
- [101] Zafeirios Fountas, Martin Benfeghoul, Adnan Omerjee, Fenia Christopoulou, Gerasimos Lampouras, Haitham Bou Ammar, and Jun Wang. Human-Inspired Episodic Memory for Infinite Context LLMs. In *International Conference on Learning Representations*, 2025. 20, 22
- [102] Noah Frazier-Logue and Stephen José Hanson. The Stochastic Delta Rule: Faster and More Accurate Deep Learning Through Adaptive Weight Noise. *Neural Computation*, 2020. 46
- [103] Noah Frazier-Logue and Stephen José Hanson. Dropout Is a Special Case of the Stochastic Delta Rule: Faster and More Accurate Deep Learning. Preprint arXiv:1808.03578, 2019. 46
- [104] Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning. In *International Conference on Learning Representations*, 2025. 9, 20, 21
- [105] Kunihiko Fukushima. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. *IEEE Transactions on Systems Science and Cybernetics*, 1969. 6, 9, 45, 47

- [106] Kuniyiko Fukushima. Cognitron: A Self-Organizing Multilayered Neural Network. *Biological Cybernetics*, 1975. 6, 45
- [107] Philip Gage. A New Algorithm for Data Compression. *The C Users Journal*, 1994. 9, 58
- [108] Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can Looped Transformers Learn to Implement Multi-Step Gradient Descent for In-Context Learning? In *International Conference on Machine Learning*, 2024. 66
- [109] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. In *International Conference on Learning Representations*, 2024. 9, 19, 20
- [110] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional Sequence to Sequence Learning. In *International Conference on Machine Learning*, 2017. 11, 52
- [111] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kaillkhura, Abhinav Bhatele, and Tom Goldstein. Scaling Up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. In *Neural Information Processing Systems*, 2025. 7, 9, 65, 66
- [112] Gemma Team, Google DeepMind. Gemma 2: Improving Open Language Models at a Practical Size. Preprint arXiv:2408.00118, 2024. 68, 75
- [113] Gemma Team, Google DeepMind. Gemma 4 Technical Report. Preprint arXiv:2607.02770, 2026. 75, 78
- [114] Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On Training Implicit Models. In *Neural Information Processing Systems*, 2021. 66
- [115] Anastasios Gerontopoulos, Spyros Gidaris, and Nikos Komodakis. Multi-Token Prediction Needs Registers. In *Neural Information Processing Systems*, 2025. 61
- [116] Felix A Gers and E Schmidhuber. LSTM Recurrent Networks Learn Simple Context-Free and Context-Sensitive Languages. *IEEE Transactions on Neural Networks*, 2001. 29
- [117] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 2000. 6, 7, 11, 30, 31, 33, 36, 46
- [118] Alexander Gerstenberger, Kazuki Irie, Pavel Golik, and Hermann Ney. Domain Robust, Fast, and Compact Neural Language Models. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020. 44
- [119] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer Feed-Forward Layers Are Key-Value Memories. In *Empirical Methods in Natural Language Processing*, 2021. 45
- [120] Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped Transformers as Programmable Computers. In *International Conference on Machine Learning*, 2023. 66
- [121] Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. Multilingual Language Processing From Bytes. In *North American Chapter of the Association for Computational Linguistics*, 2016. 58
- [122] GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren,

- Jiapeng Wang, Jiabin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhao Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. GLM-5: From Vibe Coding to Agentic Engineering. Preprint arXiv:2602.15763, 2026. 23, 39, 69, 78
- [123] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & Faster Large Language Models via Multi-Token Prediction. In *International Conference on Machine Learning*, 2024. 7, 9, 60, 61, 63, 64
- [124] Olga Golovneva, Tianlu Wang, Jason Weston, and Sainbayar Sukhbaatar. Contextual Position Encoding: Learning to Count What’s Important. Preprint arXiv:2405.18719, 2024. 9, 53, 54, 57
- [125] Irving J Good. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, 1953. 10
- [126] Google DeepMind. Gemma 3 Technical Report. Preprint arXiv:2503.19786, 2025. 48, 49, 68, 75
- [127] Anand Gopalakrishnan, Robert Csordás, Jürgen Schmidhuber, and Michael C Mozer. Decoupling the “What” and “Where” With Polar Coordinate Positional Embeddings. In *International Conference on Machine Learning*, 2026. 6, 9, 53, 55
- [128] Alex Graves. Sequence Transduction with Recurrent Neural Networks. Preprint arXiv:1211.3711, 2012. 11, 29
- [129] Alex Graves. Generating Sequences with Recurrent Neural Networks. Preprint arXiv:1308.0850, 2013. 57
- [130] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. Preprint arXiv:1603.08983, 2016. 66
- [131] Riccardo Grazi, Julien Siems, Jörg K.H. Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues. In *International Conference on Learning Representations*, 2025. 6, 32, 33, 35, 38
- [132] Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, William Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah Smith, and Hannaneh Hajishirzi. OLMo: Accelerating the Science of Language Models. In *Association for Computational Linguistics*, 2024. 9, 48, 49
- [133] Stephen Grossberg. Some Networks That Can Learn, Remember, and Reproduce Any Number of Complicated Space-Time Patterns, I. *Journal of Mathematics and Mechanics*, 1969. 11, 30
- [134] Stephen T Grossberg. *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition, and Motor Control*. Boston Studies in the Philosophy of Science. D. Reidel, Dordrecht, 1982. 30
- [135] Albert Gu. *Modeling Sequences with Structured State Spaces*. Stanford University, 2023. 6, 32
- [136] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *Conference on Language Modeling*, 2024. 6, 9, 31, 33, 35, 36, 37, 38

- [137] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent Memory with Optimal Polynomial Projections. In *Neural Information Processing Systems*, 2020. 32
- [138] Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the Parameterization and Initialization of Diagonal State Space Models. In *Neural Information Processing Systems*, 2022. 33
- [139] Albert Gu, Karan Goel, and Christopher Re. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*, 2022. 31, 32, 33
- [140] Han Guo, Songlin Yang, Tarushii Goel, Eric P. Xing, Tri Dao, and Yoon Kim. Log-Linear Attention. In *International Conference on Learning Representations*, 2026. 40
- [141] Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal State Spaces Are as Effective as Structured State Spaces. In *Neural Information Processing Systems*, 2022. 33
- [142] Murilo Gustineli. A Survey on Recently Proposed Activation Functions for Deep Learning. Preprint arXiv:2204.02921, 2022. 6, 7, 45
- [143] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. 12
- [144] John Hampshire and Alex Waibel. Connectionist Architectures for Multi-Speaker Phoneme Recognition. In *Neural Information Processing Systems*, 1989. 6, 40
- [145] Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-Infinite: Zero-Shot Extreme Length Generalization for Large Language Models. In *North American Chapter of the Association for Computational Linguistics*, 2024. 19, 20
- [146] Stephen José Hanson. A Stochastic Version of the Delta Rule. *Physica D*, 1990. 46
- [147] LI Haoyang, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, HU Nicole, Wei Dong, Li Qing, and Lei Chen. A Survey on Large Language Model Acceleration Based on KV Cache Management. *Transactions on Machine Learning Research*, 2025. 5, 7, 17
- [148] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid Structural State-Space Models. In *International Conference on Learning Representations*, 2023. 33
- [149] Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling Erase and Write in Linear Attention. Preprint arXiv:2605.22791, 2026. 32
- [150] Lukas Hauzenberger, Niklas Schmidinger, Anamaria-Roberta Hartl, David Stap, Thomas Schmied, Sebastian Böck, Günter Klambauer, and Sepp Hochreiter. KVpop: Key-Value Cache Compression with Predictive Online Pruning. Preprint arXiv:2607.05061, 2026. 20, 21
- [151] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *International Conference on Computer Vision*, 2015. 45
- [152] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016. 6, 11, 47
- [153] Xu Owen He. Mixture of A Million Experts. Preprint arXiv:2407.04153, 2024. 9, 40, 44
- [154] Zhihao He, Hang Yu, Zi Gong, Shizhan Liu, Jianguo Li, and Weiyao Lin. Rodimus*: Breaking the Accuracy-Efficiency Trade-Off with Efficient Attentions. In *International Conference on Learning Representations*, 2025. 9
- [155] Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal Recurrent Neural Networks with Scaled Cayley Transform. In *International Conference on Machine Learning*, 2018. 31
- [156] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). Preprint arXiv:1606.08415, 2016. 6, 9, 46, 47

- [157] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*, 2021. 4, 5, 8
- [158] Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-Key Normalization for Transformers. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020. 9, 48, 49
- [159] Gyuryang Heo, Timothy Ngotiaoco, Kazuki Irie, Samuel J. Gershman, and Bernardo L. Sabatini. Why Depth Matters in Parallelizable Sequence Models: A Lie Algebraic View. Preprint arXiv:2603.05573, 2026. 39
- [160] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991. 46
- [161] Wolfgang Hilberg. Karl Steinbuch, ein zu Unrecht vergessener Pionier der künstlichen neuronalen Systeme. *Frequenz*, 1995. 11
- [162] Geoffrey E Hinton. Learning Distributed Representations of Concepts. In *Cognitive Science Society*, 1986. 10
- [163] Geoffrey E Hinton and David C Plaut. Using Fast Weights to Deblur Old Memories. In *Cognitive Science Society*, 1987. 11, 31
- [164] S Hochreiter, Y Bengio, P Frasconi, and J Schmidhuber. Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*, 2001. 11
- [165] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Technische Universität München, 1991. 11, 30
- [166] Sepp Hochreiter and Jürgen Schmidhuber. LSTM Can Solve Hard Long Time Lag Problems. In *Neural Information Processing Systems*, 1997. 5, 11, 47
- [167] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997. 5, 6, 7, 11, 30, 33, 47
- [168] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training Compute-Optimal Large Language Models. In *Neural Information Processing Systems*, 2022. 5, 8, 67, 72
- [169] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed Attention: Accelerating Long Context Length LLM Inference. In *Association for Computational Linguistics*, 2025. 20, 22
- [170] John J Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *National Academy of Sciences*, 1982. 11, 30
- [171] Jeremy Howard and Sebastian Ruder. Universal Language Model Fine-Tuning for Text Classification. In *Association for Computational Linguistics*, 2018. 5
- [172] Jiayi Hu, Yongqi Pan, Jusen Du, Disen Lan, Xiaqiang Tang, Qingsong Wen, Yuxuan Liang, and Weigao Sun. Comba: Improving Bilinear RNNs with Closed-Loop Control. In *Neural Information Processing Systems*, 2025. 32
- [173] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies. In *Conference on Language Modeling*, 2024. 68, 74, 76
- [174] Xiang Hu, Xinyu Wei, Hao Gu, Minshen Zhang, Tian Liang, Huayang Li, Lei Zhu, Yan Wang, Sirui Han, Yushi Bai, Kewei Tu, Haitao Mi, and Leo Liang. Hierarchical Sparse Attention Done Right: Toward Infinite Context Modeling. Preprint arXiv:2607.02980, 2026. 20, 23

- [175] Ermo Hua, Che Jiang, Xingtai Lv, Kaiyan Zhang, Ning Ding, Youbang Sun, Biqing Qi, Yuchen Fan, Xue Kai Zhu, and Bowen Zhou. Fourier Position Embedding: Enhancing Attention’s Periodic Extension for Length Generalization. In *International Conference on Machine Learning*, 2025. 9, 55
- [176] Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Over-Tokenized Transformer: Vocabulary Is Generally Worth Scaling. In *International Conference on Machine Learning*, 2025. 7, 9, 58, 59
- [177] Kaixuan Huang, Xudong Guo, and Mengdi Wang. SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths. In *Conference on Language Modeling*, 2025. 9, 63
- [178] Wei Huang, Yue Liao, Jianhui Liu, Ruifei He, Haoru Tan, Shiming Zhang, Hongsheng Li, Si Liu, and Xiaojuan Qi. Mixture Compressor for Mixture-of-Experts LLMs Gains More. In *International Conference on Learning Representations*, 2025. 9, 44
- [179] Zihao Huang, Qiyang Min, Hongzhi Huang, Yutao Zeng, Defa Zhu, Ran Guo, and Zhou Xun. Ultra-Sparse Memory Network. In *International Conference on Learning Representations*, 2025. 44
- [180] Zihao Huang, Yu Bao, Qiyang Min, Siyan Chen, Ran Guo, Hongzhi Huang, Defa Zhu, Yutao Zeng, Banggu Wu, Xun Zhou, and Siyuan Qiao. UltraMemV2: Memory Networks Scaling to 120B Parameters with Superior Long-Context Learning. In *International Conference on Learning Representations*, 2026. 44
- [181] Huawei openPangu Team. openPangu-R-72B-2512. <https://huggingface.co/openpangu/openPangu-R-72B-2512>, 2025. 48, 77
- [182] Huawei openPangu Team. openPangu-2.0-Flash. <https://huggingface.co/openpangu/openPangu-2.0-Flash>, 2026. 77
- [183] Sergey Ioffe. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, 2015. 7, 9, 48
- [184] Kazuki Irie. Why Are Positional Encodings Nonessential for Deep Autoregressive Transformers? Revisiting a Petroglyph. In *Findings of the Association for Computational Linguistics: ACL*, 2025. 14, 57
- [185] Kazuki Irie and Jürgen Schmidhuber. Images as Weight Matrices: Sequential Image Generation through Synaptic Learning Rules. In *International Conference on Learning Representations*, 2023. 31, 32
- [186] Kazuki Irie, Pavel Golik, Ralf Schlüter, and Hermann Ney. Investigations on Byte-Level Convolutional Neural Networks for Language Modeling in Low Resource Speech Recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017. 58
- [187] Kazuki Irie, Zhihong Lei, Ralf Schlüter, and Hermann Ney. Prediction of LSTM-RNN Full Context States as a Subtask for N-gram Feedforward Language Models. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2018. 44
- [188] Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. Language Modeling with Deep Transformers. In *Interspeech*, 2019. 14, 57
- [189] Kazuki Irie, Alexander Gerstenberger, Ralf Schlüter, and Hermann Ney. How Much Self-Attention Do We Need? Trading Attention for Feed-Forward Layers. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020. 17
- [190] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going Beyond Linear Transformers with Recurrent Fast Weight Programmers. In *Neural Information Processing Systems*, 2021. 31, 32
- [191] Kazuki Irie, Róbert Csordás, and Jürgen Schmidhuber. The Dual Form of Neural Networks Revisited: Connecting Test Time Predictions to Training Patterns via Spotlights of Attention. In *International Conference on Machine Learning*, 2022. 33

- [192] Kazuki Irie, Francesco Faccio, and Jürgen Schmidhuber. Neural Differential Equations for Learning to Program Neural Nets Through Continuous Learning Rules. In *Neural Information Processing Systems*, 2022. 31, 32
- [193] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. A Modern Self-Referential Weight Matrix That Learns to Modify Itself. In *International Conference on Machine Learning*, 2022. 31, 32
- [194] Kazuki Irie, Róbert Csordás, and Jürgen Schmidhuber. Practical Computational Power of Linear Transformers and Their Recurrent and Self-Referential Extensions. In *Empirical Methods in Natural Language Processing*, 2023. 31, 32
- [195] Kazuki Irie, Róbert Csordás, and Jürgen Schmidhuber. Metalearning Continual Learning Algorithms. *Transactions on Machine Learning Research*, 2025. 31
- [196] Kazuki Irie, Morris Yau, and Samuel J Gershman. Blending Complementary Memory Systems in Hybrid Quadratic-Linear Transformers. In *Neural Information Processing Systems*, 2025. 6
- [197] Ernst Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik*, 1925. 11, 29
- [198] Alekseĭ Grigor’evich Ivakhnenko and Valentin Grigorévich Lapa. *Cybernetic Predicting Devices*. CCM Information Corporation, 1965. 11
- [199] Alexey Grigorevich Ivakhnenko. Polynomial Theory of Complex Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1971. 11
- [200] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 1991. 6, 40
- [201] Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-Depth Looped Transformers for Latent Reasoning via Shortcut Modulation. In *International Conference on Learning Representations*, 2026. 9, 66
- [202] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011. 22
- [203] Frederick Jelinek. Continuous Speech Recognition by Statistical Methods. *Proceedings of the IEEE*, 1976. 10
- [204] Frederick Jelinek. Interpolated Estimation of Markov Source Parameters from Sparse Data. In *Workshop on Pattern Recognition in Practice, 1980*, 1980. 10
- [205] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Léo Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. Preprint arXiv:2310.06825, 2023. 67, 69, 74
- [206] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of Experts. Preprint arXiv:2401.04088, 2024. 9, 41, 43
- [207] Zixuan Jiang, Jiaqi Gu, Hanqing Zhu, and David Pan. Pre-RMSNorm and Pre-CRMSNorm Transformers: Equivalent and Efficient Pre-LN Transformers. In *Neural Information Processing Systems*, 2024. 7, 9, 48, 49
- [208] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *International Conference on Learning Representations*, 2024. 5
- [209] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. LLM Maybe LongLM: Self-Extend LLM Context Window Without Tuning. In *International Conference on Machine Learning*, 2024. 9, 56
- [210] Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. MoE++: Accelerating Mixture-of-Experts Methods with Zero-Computation Experts. In *International Conference on Learning Representations*, 2025. 9, 41, 44

- [211] Michael I Jordan and Robert A Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 1994. 6, 40
- [212] Roger David Joseph. *Contributions to Perceptron Theory*. PhD thesis, Cornell University, 1961. 11
- [213] Vinay Joshi, Prashant Laddha, Shambhavi Sinha, Om Ji Omer, and Sreenivas Subramoney. QCQA: Quality and Capacity-Aware Grouped Query Attention. Preprint arXiv:2406.10247, 2024. 9, 26
- [214] Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. PolySketchFormer: Fast Transformers via Sketching Polynomial Kernels. In *International Conference on Machine Learning*, 2024. 34
- [215] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural Machine Translation in Linear Time. Preprint arXiv:1610.10099, 2016. 11
- [216] Julie Kallini, Shikhar Murty, Christopher D Manning, Christopher Potts, and Róbert Csordás. MrT5: Dynamic Token Merging for Efficient Byte-Level Language Models. In *International Conference on Learning Representations*, 2025. 58
- [217] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. Preprint arXiv:2001.08361, 2020. 5, 8
- [218] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks, 2015. URL <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>. 57, 59
- [219] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention. In *International Conference on Machine Learning*, 2020. 6, 34, 37, 38, 39
- [220] Slava Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1987. 10
- [221] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The Impact of Positional Encoding on Length Generalization in Transformers. In *Neural Information Processing Systems*, 2024. 55
- [222] Zohaib Khan, Muhammad Khaqan, Omer Tafveez, Burhanuddin Samiwala, and Agha Ali Raza. Beyond Uniform Query Distribution: Key-Driven Grouped Query Attention. Preprint arXiv:2408.08454, 2024. 9, 26
- [223] Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-Agnostic KV Cache Compression with Context Reconstruction. In *Advances in Neural Information Processing Systems*, 2025. arXiv:2505.23416. 20, 21
- [224] Janghyeon Kim, Minsoo Kim, Kyuhong Shim, and Jungwook Choi. BeaconKV: Key-Value Cache Compression Guided by Beacon Queries for Efficient Large Reasoning Model Inference. In *International Conference on Machine Learning*, 2026. arXiv:2609.04971. 20, 21
- [225] Yoon Kim and Yacine Jernite David Sontag Alexander Rush. Character-Aware Neural Language Models. In *AAAI Conference on Artificial Intelligence*, 2016. 58
- [226] Young Jin Kim, Ammar Ahmad Awan, Alexandre Muzio, Andres Felipe Cruz Salinas, Liyang Lu, Amr Hendy, Samyam Rajbhandari, Yuxiong He, and Hany Hassan Awadalla. Scalable and Efficient MoE Training for Multitask Multilingual Models. Preprint arXiv:2109.10465, 2021. 42
- [227] John Kirchenbauer et al. Multi-Token Prediction via Self-Distillation. Preprint arXiv:2602.06019, 2026. 63
- [228] Louis Kirsch and Jürgen Schmidhuber. Meta Learning Backpropagation and Improving It. In *Neural Information Processing Systems*, 2021. 31
- [229] Louis Kirsch and Jürgen Schmidhuber. Self-Referential Meta Learning. In *First Conference on Automated Machine Learning (Late-Breaking Workshop)*, 2022. 31

- [230] Shun Kiyono, Sosuke Kobayashi, Jun Suzuki, and Kentaro Inui. SHAPE: Shifted Absolute Position Embedding for Transformers. In *Empirical Methods in Natural Language Processing*, 2021. 9, 52
- [231] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-Normalizing Neural Networks. In *Neural Information Processing Systems*, 2017. 6, 45
- [232] Stephen Cole Kleene. *Representation of Events in Nerve Nets and Finite Automata*. Princeton University Press Princeton, 1956. 11, 29, 30
- [233] Reinhard Kneser and Hermann Ney. Improved Clustering Techniques for Class-Based Statistical Language Modelling. In *Eurospeech*, 1993. 10
- [234] Reinhard Kneser and Hermann Ney. Improved Backing-Off for M-gram Language Modeling. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995. 10
- [235] Teuvo Kohonen. *Associative Memory: A System-Theoretical Approach*. Communication and Cybernetics. Springer-Verlag, 1977. 30
- [236] Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebia, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling Laws for Fine-Grained Mixture of Experts. In *International Conference on Machine Learning*, 2024. 9, 41, 43, 44, 84
- [237] Hendrik A Kramers and Gregory H Wannier. Statistics of the Two-Dimensional Ferromagnet. Part I. *Physical Review*, 1941. 11, 29
- [238] Taku Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Association for Computational Linguistics*, 2018. 58
- [239] Taku Kudo and John Richardson. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. In *Empirical Methods in Natural Language Processing*, 2018. 9, 58
- [240] Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved Sequence Modeling Using State Space Principles. In *International Conference on Learning Representations*, 2026. 33
- [241] Guillaume Lample, Alexandre Sablayrolles, Marc’Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large Memory Layers with Product Keys. In *Neural Information Processing Systems*, 2019. 43
- [242] Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Trigger-Based Language Models: A Maximum Entropy Approach. In *International Conference on Acoustics, Speech and Signal Processing*, 1993. 10
- [243] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. Preprint arXiv:1504.00941, 2015. 31
- [244] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *USENIX Symposium on Operating Systems Design and Implementation*, 2024. 20, 22
- [245] Wilhelm Lenz. Beiträge zum Verständnis der magnetischen Eigenschaften in festen Körpern. *Physikalische Zeitschrift*, 1920. 11, 29
- [246] Dmitry Lepikhin, HyukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *International Conference on Learning Representations*, 2021. 6, 9, 41, 42, 43, 45
- [247] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *International Conference on Machine Learning*, 2023. 7, 9, 60, 63
- [248] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE Layers: Simplifying Training of Large, Sparse Models. In *International Conference on Machine Learning*, 2021. 6, 9, 41, 42, 44

- [249] Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. MiniMax-01: Scaling Foundation Models with Lightning Attention. Preprint arXiv:2501.08313, 2025. 39, 68, 70, 77, 83
- [250] Jingyao Li, Han Shi, Sitong Wu, Chuanyang Zheng, Zhenguo Li, Xin Jiang, Hong Xu, and Jiaya Jia. QuickLLaMA: Query-Aware Inference Acceleration for Large Language Models. In *International Conference on Computational Linguistics*, 2025. 9, 20, 22
- [251] Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models. Preprint arXiv:2208.03306, 2022. 44
- [252] Pengxiang Li, Lu Yin, and Shiwei Liu. Mix-LN: Unleashing the Power of Deeper Layers by Combining Pre-LN and Post-LN. Preprint arXiv:2412.13795, 2024. 7, 9, 50, 51
- [253] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. StarCoder: May the Source Be with You! *Transactions on Machine Learning Research*, 2023. 67, 73
- [254] Shanda Li, Chong You, Guru Guruganesh, Joshua Ainslie, Santiago Ontanon, Manzil Zaheer, Sumit Sanghai, Yiming Yang, Sanjiv Kumar, and Srinadh Bhojanapalli. Functional Interpolation for Relative Positions Improves Long Context Transformers. In *International Conference on Learning Representations*, 2023. 9, 53, 54
- [255] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM Knows What You Are Looking for Before Generation. In *Neural Information Processing Systems*, 2024. 9, 19, 20
- [256] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. In *International Conference on Machine Learning*, 2024. 7, 9, 62, 63, 64
- [257] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Empirical Methods in Natural Language Processing*, 2024. 9, 62, 63, 64
- [258] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling Up Inference Acceleration of Large Language Models via Training-Time Test. In *Neural Information Processing Systems*, 2025. 9, 62, 64
- [259] Zeping Li, Xinlong Yang, Ziheng Gao, Ji Liu, et al. Amphista: Bi-Directional Multi-Head Decoding for Accelerating LLM Inference. In *North American Chapter of the Association for Computational Linguistics*, 2025. 61
- [260] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. The Lazy Neuron Phenomenon: On Emergence of Activation Sparsity in Transformers. In *International Conference on Learning Representations*, 2023. 41
- [261] Bingli Liao and Danilo Vasconcellos Vargas. Beyond KV Caching: Shared Attention for Efficient LLMs. Preprint arXiv:2407.12866, 2024. 9, 29
- [262] Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical Details and Evaluation. White paper, AI21 Labs, 2021. 67, 72

- [263] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A Hybrid Transformer-Mamba Language Model. In *International Conference on Learning Representations*, 2025. 68, 70, 76, 81, 82
- [264] Tatiana Likhomanenko, Qiantong Xu, Gabriel Synnaeve, Ronan Collobert, and Alex Rogozhnikov. CAPE: Encoding Relative Positions with Continuous Augmented Positional Embeddings. In *Neural Information Processing Systems*, 2021. 9, 52
- [265] Zhixuan Lin, Evgenii Nikishin, Xu Owen He, and Aaron Courville. Forgetting Transformer: Softmax Attention with a Forget Gate. In *International Conference on Learning Representations*, 2025. 40
- [266] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A Structured Self-Attentive Sentence Embedding. In *International Conference on Learning Representations*, 2017. 11
- [267] Ling-Team, Ang Li, Ben Liu, Binbin Hu, Bing Li, Bingwei Zeng, Borui Ye, Caizhi Tang, Changxin Tian, Chao Huang, Chao Zhang, Chen Qian, Chenchen Ju, Chenchen Li, Chengfu Tang, Chili Fu, Chunshao Ren, Chunwei Wu, Cong Zhang, Cunyin Peng, Dafeng Xu, Daixin Wang, Dalong Zhang, Dingnan Jin, Dingyuan Zhu, Dongke Hu, Fangzheng Zhao, Feifan Wu, Feng Zhu, Gangshan Wang, Haitao Zhang, Hailin Zhao, Hanxiao Zhang, Hanzi Wang, Hao Qian, Haoyi Yu, Heng Zhang, Hongliang Zhang, Hongzhi Luan, Huirong Dong, Huizhong Li, Jia Li, Jia Liu, Jialong Zhu, Jian Sha, Jianping Wei, Jiaolong Yang, Jieyue Ma, Jiewei Wu, Jinjing Huang, Jingyun Tian, Jingyuan Zhang, Jinquan Sun, Juanhui Tu, Jun Liu, Jun Xu, Jun Zhou, Junjie Ou, Junpeng Fang, Kaihong Zhang, Kaiqin Hu, Ke Shi, Kun Tang, Kunlong Chen, Lanyin Mei, Lei Liang, Lei Xu, Libo Zhang, Lin Ju, Lin Yuan, Ling Zhong, Lintao Ma, Lu Liu, Lu Yu, Lun Cai, Meiqi Zhu, Mengying Li, Min Chen, Minghao Xue, Minghong Cai, Mingming Yin, Peijie Jiang, Peilong Zhao, Pingping Liu, Qian Zhao, Qing Cui, Qingxiang Huang, Qingyuan Yang, Quankun Yu, Shaowei Wei, Shijie Lian, Shoujian Zheng, Shun Song, Shungen Zhang, Shuo Zhang, Siyuan Li, Song Liu, Ting Guo, Tong Zhao, Wanli Gu, Weichang Wu, Weiguang Han, Wenjing Fang, Wubin Wang, Xiang Shu, Xiao Shi, Xiaoshun Lan, Xiaolu Zhang, Xiaqing Sun, Xin Zhao, Xingyu Lu, Xiong Xu, Xudong Wang, Xudong Wang, Xuemin Yang, Yajie Yang, Yang Xiang, Yanzhe Li, Yi Zhang, Yilong Wang, Yingxue Li, Yongzhen Guo, Yuzhuo Fu, Yuanyuan Wang, Yue Yang, Yue Yu, Yufeng Deng, Yun Zhang, Yunfei Xu, Yuqi Zhang, Yuxiao He, Zengke Gui, Zhaoxin Huan, Zhaoyang Wang, Zhibo Zhu, Zhihao Wang, Zhiqiang Zhang, Zhoufei Wang, Zihang Zeng, Ziqi Liu, Zitao Xuan, and Zuoli Tang. Every Activation Boosted: Scaling General Reasoner to 1 Trillion Open Language Foundation. Preprint arXiv:2510.22115, 2025. 68, 78
- [268] Seppo Linnainmaa. *The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors*. PhD thesis, Master’s Thesis (in Finnish), Univ. Helsinki, 1970. 30
- [269] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 Technical Report. Preprint arXiv:2412.19437, 2024. 7, 9, 10, 43, 61, 62, 64, 68, 69, 70, 76, 82, 83
- [270] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Reza Haffari, and Bohan Zhuang. MiniCache: KV Cache Compression in Depth Dimension for Large Language Models. In *Neural Information Processing Systems*, 2024. 5, 9, 18, 28
- [271] Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and qiang liu. Longhorn: State Space Models Are Amortized Online Learners. In *International Conference on Learning Representations*, 2025. 9, 37, 38
- [272] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval. In *Neural Information Processing Systems*, 2025. 20, 22
- [273] Weiyang Liu, Yan-Ming Zhang, Xingguo Li, Zhiding Yu, Bo Dai, Tuo Zhao, and Le Song. Deep Hyperspherical Learning. In *Neural Information Processing Systems*, 2017. 9, 48, 49

- [274] Wenhao Liu, Hao Shi, Yunhe Li, Weizhi Fei, Xiangyuan Wang, Mengzhe Ruan, Hanxu Hou, Peisong Wang, Linqi Song, and Shuang Qiu. ReasonAlloc: Hierarchical Decoding-Time KV Cache Budget Allocation for Reasoning Models. Preprint arXiv:2606.11164, 2026. 20, 21
- [275] Xuanqing Liu, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. Learning to Encode Position for Transformer with Continuous Dynamical Model. In *International Conference on Machine Learning*, 2020. 9, 52
- [276] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin Transformer V2: Scaling Up Capacity and Resolution. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 49
- [277] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. In *Neural Information Processing Systems*, 2023. 19, 20
- [278] LLM-Core Xiaomi. MiMo: Unlocking the Reasoning Potential of Language Model – From Pretraining to Posttraining. Preprint arXiv:2505.07608, 2025. 7, 9, 62, 63, 64, 68, 70, 75
- [279] LLM-Core Xiaomi. MiMo-V2-Flash Technical Report. Preprint arXiv:2601.02780, 2026. 9, 62, 64, 68, 69, 70, 78
- [280] LLM Stats. Open LLM Leaderboard 2026: Ranking the Best Open LLMs by Performance, Price, and Speed. LLM Stats, 2026. URL <https://llm-stats.com/leaderboards/open-llm-leaderboard>. Accessed 2026-07-28. 4
- [281] Ilya Loshchilov, Cheng-Ping Hsieh, Simeng Sun, and Boris Ginsburg. nGPT: Normalized Transformer with Representation Learning on the Hypersphere. In *International Conference on Learning Representations*, 2025. 51
- [282] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. MoBA: Mixture of Block Attention for Long-Context LLMs. In *Neural Information Processing Systems*, 2025. 5, 9, 20, 23
- [283] Xin Ma, Yang Liu, Jingjing Liu, and Xiaoxu Ma. Mesa-Extrapolation: A Weave Position Encoding Method for Enhanced Extrapolation in LLMs. In *Neural Information Processing Systems*, 2024. 56
- [284] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the International Conference on Machine Learning (ICML), Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013. 45
- [285] Huanru Henry Mao. Fine-Tuning Pre-Trained Transformers into Decaying Fast Weights. In *Empirical Methods in Natural Language Processing*, 2022. 9, 35, 37, 38
- [286] Andreĭ Andreevich Markov. An Example of Statistical Investigation of the Text Eugene Onegin Concerning the Connection of Samples in Chains. *Science in Context*, 2006. 10
- [287] David Marr. Simple Memory: A Theory for Archicortex. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 1971. 11, 30
- [288] Eric Martin and Chris Cundy. Parallelizing Linear Recurrent Neural Nets Over Sequence Length. In *International Conference on Learning Representations*, 2018. 31
- [289] Warren S McCulloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 1943. 11, 30
- [290] Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long Range Language Modeling via Gated State Spaces. In *International Conference on Learning Representations*, 2023. 33

- [291] Meituan LongCat Team. Introducing LongCat-2.0. LongCat Official Blog, <https://longcat.chat/blog/longcat-2.0/>, 2026. Accessed: 2026-07-06. 23, 78
- [292] Meituan LongCat Team. Scaling Embeddings Outperforms Scaling Experts in Language Models. Preprint arXiv:2601.21204, 2026. 78
- [293] Fanxu Meng, Pingzhi Tang, Xiaojuan Tang, Zengwei Yao, Xing Sun, and Muhan Zhang. TransMLA: Multi-Head Latent Attention Is All You Need. Preprint arXiv:2502.07864, 2025. 9, 28
- [294] William Merrill, Jackson Petty, and Ashish Sabharwal. The Illusion of State in State-Space Models. In *International Conference on Machine Learning*, 2024. 6, 38, 39
- [295] Meta AI. Introducing Meta Llama 3: The Most Capable Openly Available LLM to Date, 2024. URL <https://ai.meta.com/blog/meta-llama-3/>. 6, 10, 46, 54, 58, 67, 68, 69, 73
- [296] Meta AI. Introducing Llama 3.1: Our Most Capable Models to Date, 2024. URL <https://ai.meta.com/blog/meta-llama-3-1/>. 10, 54, 67, 68, 73
- [297] Meta AI. The Llama 4 Herd: Architecture, Training, Evaluation, and Deployment Notes. Preprint arXiv:2601.11659, 2026. 57, 68, 77
- [298] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, et al. SpecInfer: Accelerating Large Language Model Serving with Tree-Based Speculative Inference and Verification. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024. 9, 63
- [299] Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. Between Words and Characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. Preprint arXiv:2112.10508, 2021. 7, 57
- [300] Tomas Mikolov and Geoffrey Zweig. Context Dependent Recurrent Neural Network Language Model. In *IEEE Spoken Language Technology Workshop (SLT)*, 2012. 11
- [301] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent Neural Network Based Language Model. In *Interspeech*, 2010. 11, 29, 58, 59
- [302] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Extensions of Recurrent Neural Network Language Model. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011. 11, 29
- [303] Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. Subword Language Modeling with Neural Networks. <https://www.fit.vut.cz/person/imikolov/public/rnnlm/char.pdf>, 2012. 58
- [304] MiniCPM Team. MiniCPM4: Ultra-Efficient LLMs on End Devices. Preprint arXiv:2506.07900, 2025. 68, 74
- [305] MiniMax. MiniMax-M3 Official Configuration. <https://huggingface.co/MiniMaxAI/MiniMax-M3/blob/main/config.json>, 2026. SwiGLU-OAI activation with alpha 1.702 and limit 7.0; accessed 2026-09-17. 46, 77
- [306] MiniMax AI. MiniMax-M1: Scaling Test-Time Compute Efficiently with Lightning Attention. Preprint arXiv:2506.13585, 2025. 68, 70, 77
- [307] MiniMax AI. MiniMax-M2.1. <https://huggingface.co/MiniMaxAI/MiniMax-M2.1>, 2025. 77
- [308] MiniMax AI. The MiniMax-M2 Series: Mini Activations Unleashing Max Real-World Intelligence. Preprint arXiv:2605.26494, 2026. 70, 77
- [309] MiniMax AI. MiniMax-M2.5. <https://huggingface.co/MiniMaxAI/MiniMax-M2.5>, 2026. 77
- [310] MiniMax AI. MiniMax-M2.7. <https://huggingface.co/MiniMaxAI/MiniMax-M2.7>, 2026. 77

- [311] MiniMax AI. MiniMax Sparse Attention. Preprint arXiv:2606.13392, 2026. 20, 23, 70, 77, 83
- [312] Mistral. Mixtral of Experts: A High Quality Sparse Mixture-of-Experts, 2023. URL <https://mistral.ai/news/mixtral-of-experts>. 68, 76
- [313] Mistral. Cheaper, Better, Faster, Stronger: Continuing to Push the Frontier of AI and Making It Accessible to All, 2024. URL <https://mistral.ai/news/mixtral-8x22b>. 6, 68, 76
- [314] Mistral AI. Introducing Mistral 3. <https://mistral.ai/news/mistral-3>, 2025. 68, 69, 76
- [315] Moonshot AI. Kimi K2.6. <https://huggingface.co/moonshotai/Kimi-K2.6>, 2026. 77
- [316] Moonshot AI. Kimi K2.7-Code. <https://huggingface.co/moonshotai/Kimi-K2.7-Code>, 2026. 77
- [317] Moonshot AI. Kimi K3: Open Frontier Intelligence. Preprint arXiv:2607.24653, 2026. 39, 40, 44, 46, 51, 77
- [318] Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T. Konstantin Rusch, and Antonio Orvieto. Fixed-Point Reasoners: Stable and Adaptive Deep Looped Transformers. Preprint arXiv:2606.18206, 2026. 9, 65, 66
- [319] Siyuan Mu and Sen Lin. A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications. Preprint arXiv:2503.07137, 2025. 7
- [320] Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. OLMoE: Open Mixture-of-Experts Language Models. In *International Conference on Learning Representations*, 2025. 6, 43, 68, 77
- [321] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave No Context Behind: Efficient Infinite Context Transformers with Infini-Attention. Preprint arXiv:2404.07143, 2024. 6
- [322] Mohammed Muqeeth, Haokun Liu, and Colin Raffel. Soft Merging of Experts with Adaptive Routing. *Transactions on Machine Learning Research*, 2024. 41
- [323] Masami Nakamura and Kiyohiro Shikano. A Study of English Word Category Prediction Based on Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 1989. 10, 29
- [324] Kaoru Nakano. Associatron-a Model of Associative Memory. *IEEE Transactions on Systems, Man, and Cybernetics*, 1972. 11, 30
- [325] Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference. In *International Conference on Machine Learning*, 2024. 20, 21
- [326] Hermann Ney, Ute Essen, and Reinhard Kneser. On Structuring Probabilistic Dependences in Stochastic Language Modelling. *Computer Speech & Language*, 1994. 10
- [327] Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4ND: Modeling Images and Videos as Multidimensional Signals Using State Spaces. In *Neural Information Processing Systems*, 2022. 33
- [328] Toan Q Nguyen and Julian Salazar. Transformers without Tears: Improving the Normalization of Self-Attention. In *International Conference on Spoken Language Translation*, 2019. 7, 9, 48, 49, 51
- [329] William D. Nordhaus. Two Centuries of Productivity Growth in Computing. *The Journal of Economic History*, 2007. 5
- [330] NVIDIA. Nemotron 3 Nano: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning. Preprint arXiv:2512.20848, 2025. 46, 70, 78
- [331] NVIDIA. Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models. Preprint arXiv:2504.03624, 2025. 46, 68, 70, 75

- [332] NVIDIA. NVIDIA Nemotron Nano 2: An Accurate and Efficient Hybrid Mamba-Transformer Reasoning Model. Preprint arXiv:2508.14444, 2025. 68, 70, 75
- [333] NVIDIA. NVIDIA-Nemotron-3-Nano-4B Model Card. <https://huggingface.co/nvidia/NVIDIA-Nemotron-3-Nano-4B-BF16>, 2026. Dense hybrid Mamba-2/Transformer model compressed from Nemotron Nano 2 (9B); released March 2026. Accessed 2026-09-18. 75
- [334] NVIDIA. Nemotron 3 Super: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning. Preprint arXiv:2604.12374, 2026. 44, 78
- [335] NVIDIA. Nemotron 3 Ultra: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning. Preprint arXiv:2606.15007, 2026. 44, 78
- [336] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 OLMo 2 Furious. In *Conference on Language Modeling*, 2025. 7, 9, 48, 49
- [337] OpenAI. GPT-OSS Reference Implementation: SwiGLU. https://github.com/openai/gpt-oss/blob/main/gpt_oss/torch/model.py, 2025. Accessed 2026-09-17. 46
- [338] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, et al. GPT-4 Technical Report. Preprint arXiv:2303.08774, 2024. 18
- [339] OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Korylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b Model Card. Preprint arXiv:2508.10925, 2025. 46, 47, 68, 78
- [340] OpenBMB. MiniCPM3-4B. <https://huggingface.co/openbmb/MiniCPM3-4B>, 2024. 68, 74
- [341] OpenBMB. MiniCPM5-1B. <https://huggingface.co/openbmb/MiniCPM5-1B>, 2026. 68, 74
- [342] OpenBMB. MiniCPM5-2B. <https://huggingface.co/openbmb/MiniCPM5-2B>, 2026. 74
- [343] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting Recurrent Neural Networks for Long Sequences. In *International Conference on Machine Learning*, 2023. 6, 9, 31, 37, 38
- [344] Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, et al. Byte Latent Transformer: Patches Scale Better Than Tokens. In *Association for Computational Linguistics*, 2025. 7, 9, 58, 59
- [345] Bowen Pan, Yikang Shen, Haokun Liu, Mayank Mishra, Gaoyuan Zhang, Aude Oliva, Colin Raffel, and Rameswar Panda. Dense Training, Sparse Inference: Rethinking Training of Mixture-of-Experts Language Models. Preprint arXiv:2404.05567, 2024. 41

- [346] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A Decomposable Attention Model for Natural Language Inference. In *Empirical Methods in Natural Language Processing*, 2016. 11
- [347] Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. Can Mamba Learn How to Learn? A Comparative Study on In-Context Learning Tasks. Preprint arXiv:2402.04248, 2024. 69
- [348] Jungwoo Park, Young Jin Ahn, Kee-Eung Kim, and Jaewoo Kang. Monet: Mixture of Monosemantic Experts for Transformers. In *International Conference on Learning Representations*, 2025. 9, 41, 44
- [349] Sungrae Park, Geewook Kim, Junyeop Lee, Junbum Cha, Ji-Hoon Kim, and Hwalsuk Lee. Scale Down Transformer by Grouping Features for a Lightweight Character-Level Language Model. In *International Conference on Computational Linguistics*, 2020. 58
- [350] Badri Narayana Patro and Vijay Srinivas Agneeswaran. Mamba-360: Survey of State Space Models as Transformer Alternative for Long Sequence Modelling: Methods, Applications, and Challenges. Preprint arXiv:2404.16112, 2024. 7
- [351] Romain Paulus, Caiming Xiong, and Richard Socher. A Deep Reinforced Model for Abstractive Summarization. In *International Conference on Learning Representations*, 2018. 11
- [352] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. RWKV: Reinventing RNNs for the Transformer Era. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023. 6, 31
- [353] Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, et al. Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence. In *Conference on Language Modeling*, 2024. 9, 37, 38
- [354] Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Haowen Hou, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 “Goose” with Expressive Dynamic State Evolution. In *Conference on Language Modeling*, 2025. 6, 9, 32, 33, 35, 37, 38
- [355] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient Context Window Extension of Large Language Models. In *International Conference on Learning Representations*, 2023. 6, 9, 56
- [356] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random Feature Attention. In *International Conference on Learning Representations*, 2021. 9, 34, 37, 38
- [357] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep Contextualized Word Representations. In *North American Chapter of the Association for Computational Linguistics*, 2018. 5, 12
- [358] Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. Language Model Tokenizers Introduce Unfairness Between Languages. In *Neural Information Processing Systems*, 2024. 7, 57
- [359] Piotr Piękos, Róbert Csordás, and Jürgen Schmidhuber. Mixture of Sparse Attention: Content-Based Learnable Sparse Attention via Expert-Choice Routing. *Transactions on Machine Learning Research*, 2026. 9, 20, 23, 43
- [360] Jordan B Pollack. Recursive Distributed Representations. *Artificial Intelligence*, 1990. 10
- [361] Ofir Press, Noah Smith, and Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *International Conference on Learning Representations*, 2021. 9, 53, 54, 55, 67
- [362] Weizhen Qi, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-Training. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020. 7, 9, 61, 64

- [363] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosFormer: Rethinking Softmax In Attention. In *International Conference on Learning Representations*, 2022. 34
- [364] Zhen Qin, Weigao Sun, Dong Li, Xuyang Shen, Weixuan Sun, and Yiran Zhong. Lightning Attention-2: A Free Lunch for Handling Unlimited Sequence Lengths in Large Language Models. Preprint arXiv:2401.04658, 2024. 39
- [365] Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. HGRN2: Gated Linear RNNs with State Expansion. In *Conference on Language Modeling*, 2024. 9, 37, 38
- [366] Zhen Qin, Yuxin Mao, Xuyang Shen, Dong Li, Jing Zhang, Yuchao Dai, and Yiran Zhong. You Only Scan Once: Efficient Multi-Dimension Sequential Modeling with LightNet. *Transactions on Machine Learning Research*, 2026. 40
- [367] Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. CAKE: Cascading and Adaptive KV Cache Eviction with Layer Preferences. In *International Conference on Learning Representations*, 2025. 19, 20
- [368] Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated Attention for Large Language Models: Non-Linearity, Sparsity, and Attention-Sink-Free. In *Neural Information Processing Systems*, 2025. 40, 47, 48
- [369] Haohao Qu, Liangbo Ning, Rui An, Wenqi Fan, Tyler Derr, Hui Liu, Xin Xu, and Qing Li. A Survey of Mamba. Preprint arXiv:2408.01129, 2024. 7
- [370] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report. Preprint arXiv:2412.15115, 2025. 10, 67, 69, 74
- [371] Qwen Team. Qwen3 Technical Report. Preprint arXiv:2505.09388, 2025. 10, 48, 49, 58, 68, 74, 77
- [372] Qwen Team. Qwen3.5: Towards Native Multimodal Agents, 2026. URL <https://qwen.ai/blog?id=qwen3.5>. 39, 74, 77
- [373] Qwen Team. Qwen3.6-Plus: Towards Real World Agents, 2026. URL <https://qwen.ai/blog?id=qwen3.6>. 39, 74, 77
- [374] Qwen Team. Qwen3.6-27B: Flagship-Level Coding in a 27B Dense Model. Qwen Blog, April 2026. URL <https://qwen.ai/blog?id=qwen3.6-27b>. Accessed 2026-07-29. 4
- [375] Qwen Team. Qwen3.8-2.4T-A95B, 2026. URL <https://huggingface.co/Qwen/Qwen3.8-2.4T-A95B>. 77
- [376] Qwen Team. Qwen3.8-27B, 2026. URL <https://huggingface.co/Qwen/Qwen3.8-27B>. 74
- [377] Qwen Team. Qwen3.8-Max: A New Bar for Coding and Cowork, 2026. URL https://www.alibabacloud.com/blog/qwen3-8-max-a-new-bar-for-coding-and-cowork_603421. 77
- [378] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. Technical report, OpenAI, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf. 6, 12, 16, 52
- [379] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models Are Unsupervised Multitask Learners. OpenAI Blog, 2019. 12, 49, 51, 52, 58, 59
- [380] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. Preprint arXiv:2112.11446, 2021. 67, 72

- [381] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 2020. 9, 16, 53, 54, 67, 72
- [382] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. In *International Conference on Machine Learning*, 2022. 6, 9, 41, 43, 45
- [383] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. Preprint arXiv:1710.05941, 2017. 6, 46, 47
- [384] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-Depths: Dynamically Allocating Compute in Transformer-Based Language Models. Preprint arXiv:2404.02258, 2024. 7, 9, 43, 44, 65
- [385] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In *Conference on Language Modeling*, 2024. 5
- [386] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ Attention: Bandwidth-Efficient LLM Inference. In *International Conference on Machine Learning*, 2024. 20, 22
- [387] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling Vision with Sparse Mixture of Experts. In *Neural Information Processing Systems*, 2021. 42
- [388] Anthony J Robinson. An Application of Recurrent Nets to Phone Probability Estimation. *IEEE Transactions on Neural Networks*, 1994. 11, 29
- [389] Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash Layers For Large Sparse Models. In *Neural Information Processing Systems*, 2021. 41, 44
- [390] Frank Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 1958. 11
- [391] Frank Rosenblatt et al. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan books Washington, DC, 1962. 11
- [392] Ronald Rosenfeld. Two Decades of Statistical Language Modeling: Where Do We Go from Here? *Proceedings of the IEEE*, 2002. 10
- [393] D. E. Rumelhart and J. L. McClelland. On Learning the Past Tenses of English Verbs. In *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*. MIT Press, 1986. 10
- [394] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning Internal Representations by Error Propagation. ICS Report 8506, Institute for Cognitive Science, University of California, San Diego, 1985. 30
- [395] Tim Salimans and Durk P Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Neural Information Processing Systems*, 2016. 9, 48, 49
- [396] Alexander Samarin, Sergei Krutikov, Anton Shevtsov, Sergei Skvortsov, Filipp Fisin, and Alexander Golubev. LK Losses: Direct Acceptance Rate Optimization for Speculative Decoding. In *International Conference on Machine Learning*, 2026. 9, 63
- [397] Mohammad Samragh et al. Your LLM Knows the Future: Uncovering Its Multi-Token Prediction Potential. Preprint arXiv:2507.11851, 2025. 61
- [398] Yash Sarrof, Yana Veitsman, and Michael Hahn. The Expressive Capacity of State Space Models: A Formal Language Perspective. In *Neural Information Processing Systems*, 2024. 33

- [399] Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with Latent Thoughts: On the Power of Looped Transformers. In *International Conference on Learning Representations*, 2025. 66
- [400] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks. In *International Conference on Learning Representations*, 2014. 31
- [401] Imanol Schlag and Jürgen Schmidhuber. Gated Fast Weights for On-the-Fly Neural Program Generation. In *NIPS Metalearning Workshop*, 2017. 31
- [402] Imanol Schlag and Jürgen Schmidhuber. Learning to Reason with Third Order Tensor Products. In *Neural Information Processing Systems*, 2018. 31
- [403] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers. In *International Conference on Machine Learning*, 2021. 6, 9, 31, 32, 36, 37, 38, 39
- [404] Imanol Schlag, Tsendsuren Munkhdalai, and Jürgen Schmidhuber. Learning Associative Inference Using Fast Weight Memory. In *International Conference on Learning Representations*, 2021. 31
- [405] Juergen Schmidhuber. Annotated History of Modern AI and Deep Learning. Preprint arXiv:2212.11279, 2022. 11, 29, 30
- [406] Jürgen Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Recurrent Nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, 1991. 5, 6, 9, 11, 12, 31, 32, 33, 35, 36, 37, 38
- [407] Jürgen Schmidhuber. Steps Towards ‘Self-Referential’ Neural Learning: A Thought Experiment. Technical Report CU-CS-627-92, University of Colorado at Boulder, 1992. 31
- [408] Jürgen Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 1992. 5, 6, 11, 12, 31, 32, 33, 38
- [409] Jürgen Schmidhuber. Learning Complex, Extended Sequences Using the Principle of History Compression. *Neural Computation*, 1992. 54
- [410] Jürgen Schmidhuber. An ‘introspective’ network That Can Learn to Run Its Own Weight Change Algorithm. In *1993 Third International Conference on Artificial Neural Networks*, 1993. 31
- [411] Jürgen Schmidhuber. A Neural Network That Embeds Its Own Meta-Levels. In *IEEE International Conference on Neural Networks*, 1993. 31
- [412] Jürgen Schmidhuber. Reducing the Ratio Between Learning Complexity and Number of Time Varying Variables in Fully Recurrent Nets. In *International Conference on Artificial Neural Networks*, 1993. 12, 31
- [413] Jürgen Schmidhuber. A ‘Self-Referential’ Weight Matrix. In *International Conference on Artificial Neural Networks (ICANN)*, 1993. 31
- [414] Jürgen Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 2015. 11, 29
- [415] Jürgen Schmidhuber. Deep Learning: Our Miraculous Year 1990-1991. Preprint arXiv:2005.05744, 2020. 6
- [416] Jürgen Schmidhuber. 26 March 1991: Neural Nets Learn to Program Neural Nets with Fast Weights—the First Transformer Variants. 2021-: New Stuff!, 2021. URL <https://people.idsia.ch/~juergen/fast-weight-programmer-1991-transformer.html>. AI Blog. 6, 11
- [417] Jürgen Schmidhuber. Scientific Integrity and the History of Deep Learning: The 2021 Turing Lecture, and the 2018 Turing Award. Technical Report IDSIA-77-21 (v3), IDSIA, Lugano, Switzerland, 2022. 29
- [418] Jürgen Schmidhuber. Who Invented Deep Residual Learning? Preprint arXiv:2509.24732, 2025. 6, 30

- [419] Jürgen Schmidhuber. Who Invented Transformer Neural Networks? Technical Note IDSIA-11-25, IDSIA, 2025. URL <https://people.idsia.ch/~juergen/who-invented-transformer-neural-networks.html>. 5, 33
- [420] Jürgen Schmidhuber. The 1991 Unnormalized Linear Transformer (ULTRA), 2025. URL <https://people.idsia.ch/~juergen/1991-unnormalized-linear-transformer.html>. AI Blog. 5, 32, 33
- [421] Jürgen Schmidhuber and Stefan Heil. Predictive Coding with Neural Nets: Application to Text Compression. In *Neural Information Processing Systems*, 1994. 29
- [422] Jürgen Schmidhuber and Stefan Heil. Sequential Neural Text Compression. *IEEE Transactions on Neural Networks*, 1996. 10, 11, 57
- [423] Simon Schug, Seijin Kobayashi, Yassir Akram, João Sacramento, and Razvan Pascanu. Attention as a Hypernetwork. In *International Conference on Learning Representations*, 2025. 12
- [424] Mike Schuster and Kaisuke Nakajima. Japanese and Korean Voice Search. In *International Conference on Acoustics, Speech and Signal Processing*, 2012. 9, 58
- [425] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident Adaptive Language Modeling. In *Neural Information Processing Systems*, 2022. 7, 9, 64, 65
- [426] Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How Much Is One Recurrence Worth? Iso-Depth Scaling Laws for Looped Language Models. Preprint arXiv:2604.21106, 2026. 7, 9, 66
- [427] Terrence J Sejnowski and Charles R Rosenberg. NETalk: A Parallel Network That Learns to Read Aloud. In *Neurocomputing: Foundations of Research*. MIT Press, 1988. 10
- [428] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Association for Computational Linguistics*, 2016. 58
- [429] Claude E Shannon. Prediction and Entropy of Printed English. *The Bell System Technical Journal*, 1951. 10
- [430] Claude Elwood Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 1948. 10
- [431] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-Attention with Relative Position Representations. In *North American Chapter of the Association for Computational Linguistics*, 2018. 6, 54
- [432] Noam Shazeer. Fast Transformer Decoding: One Write-Head Is All You Need. Preprint arXiv:1911.02150, 2019. 5, 9, 17, 26
- [433] Noam Shazeer. GLU Variants Improve Transformer. Preprint arXiv:2002.05202, 2020. 6, 9, 45, 46, 47
- [434] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*, 2017. 6, 9, 40, 41, 44
- [435] Yikang Shen, Zhen Guo, Tianle Cai, and Zengyi Qin. JetMoE: Reaching Llama2 Performance with 0.1M Dollars. Preprint arXiv:2404.07413, 2024. 6, 68, 76
- [436] Neil Shephard and Michael K Pitt. Likelihood Analysis of Non-Gaussian Measurement Time Series. *Biometrika*, 1997. 32, 36
- [437] Weijia Shi, Akshita Bhagia, Kevin Farhat, Niklas Muennighoff, Pete Walsh, Jacob Morrison, Dustin Schwenk, Shayne Longpre, Jake Poznanski, Allyson Ettinger, et al. FlexOlm: Open Language Models for Flexible Data Use. In *Neural Information Processing Systems*, 2025. 9, 44
- [438] Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazi. DeltaProduct: Improving State-Tracking in Linear RNNs via Householder Products. In *Neural Information Processing Systems*, 2025. 32

- [439] Jimmy TH Smith, Andrew Warrington, and Scott Linderman. Simplified State Space Layers for Sequence Modeling. In *International Conference on Learning Representations*, 2023. 33
- [440] Snowflake AI Research Team. Snowflake Arctic: The Best LLM for Enterprise AI — Efficiently Intelligent, Truly Open, 2024. 9, 44
- [441] David R. So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V. Le. Primer: Searching for Efficient Transformers for Language Modeling. In *Neural Information Processing Systems*, 2021. 9, 46, 47
- [442] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014. 46
- [443] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training Very Deep Networks. In *Neural Information Processing Systems*, 2015. 6, 47
- [444] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway Networks. Preprint arXiv:1505.00387, 2015. 6, 11, 47
- [445] Karl Steinbuch. Adaptive Networks Using Learning Matrices. *Kybernetik*, 1965. 11
- [446] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise Parallel Decoding for Deep Autoregressive Models. In *Neural Information Processing Systems*, 2018. 7, 9, 60, 61, 63, 64
- [447] Jianlin Su. Time and Space: Viewing Attention as an RNN with Quadratic Complexity. Scientific Spaces (blog), March 2024. URL <https://spaces.ac.cn/archives/10017>. In Chinese. 40
- [448] Jianlin Su. Entries of DeltaNet’s Core Inverse Matrix Always Lie in $[-1, 1]$. Scientific Spaces (blog), January 2026. URL <https://spaces.ac.cn/archives/11563>. In Chinese. 40
- [449] Jianlin Su. Linearizing Softmax Attention into Gated DeltaNet. Scientific Spaces (blog), July 2026. URL <https://spaces.ac.cn/archives/11823>. In Chinese. 40
- [450] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced Transformer with Rotary Position Embedding. *Neurocomputing*, 2024. 6, 9, 27, 53, 54
- [451] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-To-End Memory Networks. In *Neural Information Processing Systems*, 2015. 44
- [452] Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen-tau Yih, Jason Weston, and Xian Li. Branch-Train-Mix: Mixing Expert LLMs into a Mixture-of-Experts LLM. In *Conference on Language Modeling*, 2024. 9, 44
- [453] Haohai Sun and MiniMax AI. Why Did M2 End Up as a Full Attention Model? MiniMax News, <https://www.minimax.io/news/why-did-m2-end-up-as-a-full-attention-model>, October 2025. 70, 83
- [454] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (Learn at Test Time): RNNs with Expressive Hidden States. In *International Conference on Machine Learning*, 2025. 32, 38
- [455] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive Network: A Successor to Transformer for Large Language Models. Preprint arXiv:2307.08621, 2023. 9, 35, 37, 38, 39
- [456] Yutao Sun, Li Dong, Barun Patra, Shuming Ma, Shaohan Huang, Alon Benhaim, Vishrav Chaudhary, Xia Song, and Furu Wei. A Length-Extrapolatable Transformer. In *Association for Computational Linguistics*, 2023. 9, 55
- [457] Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You Only Cache Once: Decoder-Decoder Architectures for Language Models. In *Neural Information Processing Systems*, 2024. 29

- [458] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM Neural Networks for Language Modeling. In *Interspeech*, 2012. 11
- [459] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating Text with Recurrent Neural Networks. In *International Conference on Machine Learning*, 2011. 57
- [460] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In *Neural Information Processing Systems*, 2014. 11, 29
- [461] Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Danning Ke, Shikuan Hong, Yiwu Yao, and Gongyi Wang. RazorAttention: Efficient KV Cache Compression through Retrieval Heads. In *International Conference on Learning Representations*, 2025. 20, 21
- [462] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. In *International Conference on Machine Learning*, 2024. 9, 20, 22
- [463] Yehui Tang, Yichun Su, Yujie Zhu, et al. Pangu Ultra MoE: How to Train Your Big MoE on Ascend NPUs. Preprint arXiv:2505.04519, 2025. 62, 68, 69, 77
- [464] Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling Laws with Vocabulary: Larger Models Deserve Larger Vocabularies. In *Neural Information Processing Systems*, 2024. 7, 59
- [465] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient Transformers: A Survey. *ACM Computing Surveys*, 2022. 5, 7
- [466] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A Large Language Model for Science. Preprint arXiv:2211.09085, 2022. 73
- [467] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open Models Based on Gemini Research and Technology. Preprint arXiv:2403.08295, 2024. 49, 58, 68, 75
- [468] Jamba Team, Barak Lenz, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, Dan Padnos, et al. Jamba-1.5: Hybrid Transformer-Mamba Models at Scale. Preprint arXiv:2408.12570, 2024. 70, 82
- [469] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi K2: Open Agentic Intelligence. Preprint arXiv:2507.20534, 2025. 68, 69, 77
- [470] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi K2.5: Scaling Reinforcement Learning with LLMs. Preprint arXiv:2602.02276, 2025. 77
- [471] Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiayi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi Linear: An Expressive, Efficient Attention Architecture. Preprint arXiv:2510.26692, 2025. 32, 39, 40, 51, 68, 70, 77
- [472] Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, et al. Attention Residuals. Preprint arXiv:2603.15031, 2026. 50
- [473] Ling Team, Anqi Shen, Baihui Li, Bin Hu, Bin Jing, Cai Chen, Chao Huang, Chao Zhang, Chaokun Yang, Cheng Lin, Chengyao Wen, Congqi Li, Deng Zhao, Dingbo Yuan, Donghai You, Fagui Mao, Fanzhuang Meng, Feng Xu, Guojie Li, Guowei Wang, Hao Dai, Haonan Zheng, Hong Liu, Jia Guo, Jiaming Liu, Jian Liu, Jianhao Fu, Jiannan Shi, Jianwen Wang, Jianxin Lai, Jin Yang, Jun Mei, Jun Zhou, Junbo Zhao, Junping Zhao, Kuan Xu, Le Su, Lei Chen, Li Tang, Liang Jiang, Liangcheng Fu, Lianhao Xu, Linfeng Shi, Lisha Liao, Longfei

- Zheng, Meng Li, Mingchun Chen, Qi Zuo, Qiang Cheng, Qianggang Cao, Qitao Shi, Quanrui Guo, Senlin Zhu, Shaofei Wang, Shaomian Zheng, Shuaicheng Li, Shuwei Gu, Siba Chen, Tao Wu, Tao Zhang, Tianyu Zhang, Tianyu Zhou, Tiwei Bie, Tongkai Yang, Wang Hong, Wang Ren, Weihua Chen, Wenbo Yu, Wengang Zheng, Xiangchun Wang, Xiaodong Yan, Xiaopei Wan, Xin Zhao, Xinyu Kong, Xinyu Tang, Xudong Han, Xudong Wang, Xuemin Yang, Xueyu Hu, Yalin Zhang, Yan Sun, Yicheng Shan, Yilong Wang, Yingying Xu, Yongkang Liu, Yongzhen Guo, Yuanyuan Wang, Yuchen Yan, Yuefan Wang, Yuhong Guo, Zehuan Li, Zhankai Xu, Zhe Li, Zhenduo Zhang, Zhengke Gui, Zhenxuan Pan, Zhenyu Huang, Zhenzhong Lan, Zhiqiang Ding, Zhiqiang Zhang, Zhixun Li, Zhizhen Liu, Zihao Wang, and Zujie Wen. Every Step Evolves: Scaling Reinforcement Learning for Trillion-Scale Thinking Model. Preprint arXiv:2510.18855, 2025. 78
- [474] Meituan LongCat Team, Bayan, Bei Li, Bingye Lei, Bo Wang, Bolin Rong, Chao Wang, Chao Zhang, Chen Gao, Chen Zhang, Cheng Sun, Chengcheng Han, Chenguang Xi, Chi Zhang, Chong Peng, Chuan Qin, Chuyu Zhang, Cong Chen, Congkui Wang, Dan Ma, Daoru Pan, Defei Bu, Dengchang Zhao, Deyang Kong, Dishan Liu, Feiye Huo, Fengcun Li, Fubao Zhang, Gan Dong, Gang Liu, Gang Xu, Ge Li, Guoqiang Tan, Guoyuan Lin, Haihang Jing, Haomin Fu, Haonan Yan, Haoxing Wen, Haozhe Zhao, Hong Liu, Hongmei Shi, Hongyan Hao, Hongyin Tang, Huantian Lv, Hui Su, Jiacheng Li, Jiahao Liu, Jiahuan Li, Jiajun Yang, Jiaming Wang, Jian Yang, Jianchao Tan, Jiaqi Sun, Jiaqi Zhang, Jiawei Fu, Jiawei Yang, Jiaxi Hu, Jiayu Qin, Jingang Wang, Jiyuan He, Jun Kuang, Junhui Mei, Kai Liang, Ke He, Kefeng Zhang, Keheng Wang, Keqing He, Liang Gao, Liang Shi, Lianhui Ma, Lin Qiu, Lingbin Kong, Lingtong Si, Linkun Lyu, Linsen Guo, Liqi Yang, Lizhi Yan, Mai Xia, Man Gao, Manyuan Zhang, Meng Zhou, Mengxia Shen, Mingxiang Tuo, Mingyang Zhu, Peiguang Li, Peng Pei, Peng Zhao, Pengcheng Jia, Pingwei Sun, Qi Gu, Qianyun Li, Qingyuan Li, Qiong Huang, Qiyuan Duan, Ran Meng, Rongxiang Weng, Ruichen Shao, Rumei Li, Shizhe Wu, Shuai Liang, Shuo Wang, Suogui Dang, Tao Fang, Tao Li, Tefeng Chen, Tianhao Bai, Tianhao Zhou, Tingwen Xie, Wei He, Wei Huang, Wei Liu, Wei Shi, Wei Wang, Wei Wu, Weikang Zhao, Wen Zan, Wenjie Shi, Xi Nan, Xi Su, Xiang Li, Xiang Mei, Xiangyang Ji, Xiangyu Xi, Xiangzhou Huang, Xianpeng Li, Xiao Fu, Xiao Liu, Xiao Wei, Xiaodong Cai, Xiaolong Chen, Xiaoqing Liu, Xiaotong Li, Xiaowei Shi, Xiaoyu Li, Xili Wang, Xin Chen, Xing Hu, Xingyu Miao, Xinyan He, Xuemiao Zhang, Xueyuan Hao, Xuezhi Cao, Xunliang Cai, Xurui Yang, Yan Feng, Yang Bai, Yang Chen, Yang Yang, Yaqi Huo, Yerui Sun, Yifan Lu, Yifan Zhang, Yipeng Zang, Yitao Zhai, Yiyang Li, Yongjing Yin, Yongkang Lv, Yongwei Zhou, Yu Yang, Yuchen Xie, Yueqing Sun, Yuewen Zheng, Yuhuai Wei, Yulei Qian, Yunfan Liang, Yunfang Tai, Yunke Zhao, Zeyang Yu, Zhao Zhang, Zhaohua Yang, Zhenchao Zhang, Zhikang Xia, Zhiye Zou, Zhizhao Zeng, Zhongda Su, Zhuofan Chen, Zijian Zhang, Ziwen Wang, Zixu Jiang, Zizhe Zhao, Zongyu Wang, and Zunhai Su. LongCat-Flash Technical Report. Preprint arXiv:2509.01322, 2025. 9, 44, 62, 64, 68, 70, 76, 78
- [475] MosaicML NLP Team. Introducing MPT-7B: A New Standard for Open-Source, Commercially Usable LLMs, 2023. URL www.mosaicml.com/blog/mpt-7b. 67, 73
- [476] Qwen Team. Introducing Qwen1.5, 2024. URL <https://qwenlm.github.io/blog/qwen1.5/>. 10, 67, 74
- [477] Qwen Team. Qwen3-Next: Towards Ultimate Training & Inference Efficiency, 2025. URL <https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd&from=research.latest-advancements-list>. 7, 9, 39, 48, 62, 68, 70, 77
- [478] Team GLM. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. Preprint arXiv:2406.12793, 2024. 73
- [479] Team GLM. GLM-4.5: An Open Multilingual Multi-Modal Mixture-of-Experts Model. Preprint arXiv:2508.06471, 2025. 7, 9, 48, 62, 64, 68, 69, 77
- [480] Team GLM. GLM-5.2. <https://huggingface.co/zai-org/GLM-5.2>, 2026. 78
- [481] Team GLM. GLM-5.3. <https://docs.z.ai/guides/llm/glm-5.3>, 2026. 78
- [482] Tencent Hunyuan Team. Hy3. <https://hy.tencent.com/research/hy3>, 2026. 78
- [483] Tencent Hunyuan Team. Hy4 Preview. <https://huggingface.co/tencent/Hy4-preview>, 2026. 50, 78
- [484] Thinking Machines Lab. Introducing Inkling. <https://thinkingmachines.ai/news/introducing-inkling/>, 2026. 53, 54, 78

- [485] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. LaMDA: Language Models for Dialog Applications. Preprint arXiv:2201.08239, 2022. 72
- [486] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and Efficient Foundation Language Models. Preprint arXiv:2302.13971, 2023. 6, 7, 10, 49, 54, 58, 67, 68
- [487] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. Preprint arXiv:2307.09288, 2023. 10, 54, 58, 59, 67, 68, 69, 73
- [488] Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer Dissection: An Unified Understanding for Transformer’s Attention via the Lens of Kernel. In *Empirical Methods in Natural Language Processing*, 2019. 9, 34, 37, 38
- [489] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Neural Information Processing Systems*, 2017. 5, 6, 9, 11, 12, 14, 15, 33, 45, 49, 51, 52
- [490] Christoph von der Malsburg. The Correlation Theory of Brain Function. Internal Report 81-2, Max-Planck-Institut für Biophysikalische Chemie, Abteilung Neurobiologie, Göttingen, 1981. 11, 31
- [491] Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, et al. MesaNet: Sequence Modeling by Locally Optimal Test-Time Training. In *International Conference on Learning Representations*, 2026. 38
- [492] Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, et al. Efficient Large Language Models: A Survey. *Transactions on Machine Learning Research*, 2024. 7
- [493] An Wang, Xingwu Sun, Ruobing Xie, Shuaipeng Li, Jiaqi Zhu, Zhen Yang, Pinxue Zhao, JN Han, Zhanhui Kang, Di Wang, et al. HMoE: Heterogeneous Mixture of Experts for Language Modeling. Preprint arXiv:2408.10681, 2024. 9, 44
- [494] Ao Wang, Hui Chen, Jianchao Tan, Kefeng Zhang, Xunliang Cai, Zijia Lin, Jungong Han, and Guiguang Ding. PrefixKV: Adaptive Prefix KV Cache Is What Vision Instruction-Following Models Need for Efficient Generation. In *Neural Information Processing Systems*, 2025. 9, 19, 20
- [495] Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and Jakob Grue Simonsen. Encoding Word Order in Complex Embeddings. In *International Conference on Learning Representations*, 2020. 52
- [496] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. DeepNet: Scaling Transformers to 1,000 Layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. 6, 9, 49
- [497] Jie Wang, Tao Ji, Yuanbin Wu, Hang Yan, Tao Gui, Qi Zhang, Xuanjing Huang, and Xiaoling Wang. Length Generalization of Causal Transformers without Position Encoding. In *Findings of the Association for Computational Linguistics: ACL*, 2024. 57
- [498] Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-Time Regression: A Unifying Framework for Designing Sequence Models with Associative Memory. Preprint arXiv:2501.12352, 2025. 6, 38, 39
- [499] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts. Preprint arXiv:2408.15664, 2024. 9, 42
- [500] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning Deep Transformer Models for Machine Translation. In *Annual Meeting of the Association for Computational Linguistics*, 2019. 49

- [501] Shuai Wang, Yao Teng, and Limin Wang. Deep Equilibrium Object Detection and Segmentation. *Transactions on Pattern Analysis and Machine Intelligence*, 2025. 66
- [502] Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization? In *International Conference on Machine Learning*, 2022. 16
- [503] Ya Wang, Zhijian Zhuo, Yutao Zeng, Xun Zhou, Jian Yang, and Xiaoqing Li. Scale-Distribution Decoupling: Enabling Stable and Effective Training of Large Language Models. Preprint arXiv:2502.15499, 2025. 49
- [504] Zili Wang, Jiajun Chai, Lin Chen, Xiaohan Wang, Shiming Xiang, and Guojun Yin. Joint Training of Multi-Token Prediction in Reinforcement Learning via Optimal Coefficient Calibration. Preprint arXiv:2605.28184, 2026. 63
- [505] Ziteng Wang, Jun Zhu, and Jianfei Chen. ReMoE: Fully Differentiable Mixture-of-Experts with ReLU Routing. In *International Conference on Learning Representations*, 2025. 9, 42
- [506] Gregory H Wannier. The Statistical Problem in Cooperative Phenomena. *Reviews of Modern Physics*, 1945. 11, 29
- [507] Paul J Werbos. Backpropagation through Time: What It Does and How to Do It. *Proceedings of the IEEE*, 78 (10):1550–1560, 1990. 30
- [508] Bernard Widrow and Marcian E Hoff. Adaptive Switching Circuits. In *Neurocomputing: Foundations of Research*, 1988. 32
- [509] Ian H Witten and Timothy C Bell. The Zero-Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression. *IEEE Transactions on Information Theory*, 1991. 10
- [510] BigScience Workshop, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. Preprint arXiv:2211.05100, 2022. 49, 67, 73
- [511] Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alexander Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-Scale Proxies for Large-Scale Transformer Training Instabilities. In *International Conference on Learning Representations*, 2024. 48
- [512] Haoyi Wu and Kewei Tu. Layer-Condensed KV Cache for Efficient Inference of Large Language Models. In *Association for Computational Linguistics*, 2024. 9, 29
- [513] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. Preprint arXiv:1609.08144, 2016. 58
- [514] Yuxin Wu and Kaiming He. Group Normalization. In *European Conference on Computer Vision*, 2018. 51
- [515] xAI. Grok-1. <https://github.com/xai-org/grok-1>, March 2024. 6
- [516] xAI. Grok 2.5. <https://huggingface.co/xai-org/grok-2>, 2025. 78
- [517] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. In *Findings of the Association for Computational Linguistics: ACL*, 2024. 7, 60
- [518] Chaojun Xiao, Pengl Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory. In *Neural Information Processing Systems*, 2024. 9, 20, 22
- [519] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations*, 2024. 5, 9, 19, 20

- [520] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. In *International Conference on Learning Representations*, 2025. 20, 21
- [521] Xiaomi MiMo Team. MiMo-V2.5. <https://huggingface.co/collections/XiaomiMiMo/mimo-v25>, 2026. 78
- [522] Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Liang Zhao, et al. mHC: Manifold-Constrained Hyper-Connections. In *International Conference on Machine Learning*, 2026. 50
- [523] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On Layer Normalization in the Transformer Architecture. In *International Conference on Machine Learning*, 2020. 6, 7, 9, 48, 49, 51
- [524] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective Long-Context Scaling of Foundation Models. In *North American Chapter of the Association for Computational Linguistics*, 2024. 56
- [525] Fangyuan Xu, Tanya Goyal, and Eunsol Choi. RefreshKV: Updating Small KV Cache During Long-Form Generation. Preprint arXiv:2411.05787, 2024. 20, 22
- [526] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and Improving Layer Normalization. In *Advances in Neural Information Processing Systems*, 2019. 48
- [527] Kevin Xu and Issei Sato. On Expressive Power of Looped Transformers: Theoretical Analysis and Enhancement via Timestep Encoding. In *International Conference on Machine Learning*, 2025. 66
- [528] Yufei Xu, Fanxu Meng, Fan Jiang, Yuxuan Wang, Ruijie Zhou, Zhaohui Wang, Jiexi Wu, Zhixin Pan, Xiaojuan Tang, Wenjie Pei, Tongxuan Liu, Di Yin, Xing Sun, and Muhan Zhang. HISA: Efficient Hierarchical Indexing for Fine-Grained Sparse Attention. In *Conference on Language Modeling*, 2026. arXiv:2603.28458. 23
- [529] Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. ThinK: Thinner Key Cache by Query-Driven Pruning. In *International Conference on Learning Representations*, 2025. 9, 28
- [530] Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. OpenMoE: An Early Effort on Open Mixture-of-Experts Language Models. In *International Conference on Machine Learning*, 2024. 68, 76
- [531] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a Token-Free Future with Pre-Trained Byte-to-Byte Models. *Transactions of the Association for Computational Linguistics*, 2022. 7, 9, 58, 59
- [532] Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. Baichuan 2: Open Large-Scale Language Models. Preprint arXiv:2309.10305, 2023. 67, 74
- [533] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jianxin Ma, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jin Xu, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 Technical Report. Preprint arXiv:2407.10671, 2024. 56, 67, 68, 69, 74, 76
- [534] Bowen Yang, Bharat Venkitesh, Dwaraknath Gnaneshwar Talupuru, Hangyu Lin, David Cairuz, Phil Blunsom, and Acyr Locatelli. RoPE to NoPE and Back Again: A New Hybrid Attention Strategy. In *Neural Information Processing Systems*, 2025. 57

- [535] Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped Transformers Are Better at Learning Learning Algorithms. In *International Conference on Learning Representations*, 2024. 66
- [536] Qingyue Yang, Jie Wang, Xing Li, Zhihai Wang, Chen Chen, Lei Chen, Xianzhi Yu, Wulong Liu, Jianye Hao, Mingxuan Yuan, and Bin Li. AttentionPredictor: Temporal Pattern Matters for Efficient LLM Inference. In *Neural Information Processing Systems*, 2025. 20, 22
- [537] Songlin Yang. DeltaNet Explained. <https://sustcsonglin.github.io/blog/2024/deltanet-3/>, 2024. 38
- [538] Songlin Yang and Yu Zhang. FLA: A Triton-Based Library for Hardware-Efficient Implementations of Linear Attention Mechanism, January 2024. URL <https://github.com/fla-org/flash-linear-attention>. 39, 40
- [539] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated Linear Attention Transformers with Hardware-Efficient Training. In *International Conference on Machine Learning*, 2024. 9, 33, 35, 37, 38, 39
- [540] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing Linear Transformers with the Delta Rule over Sequence Length. In *Neural Information Processing Systems*, 2024. 6, 32, 35
- [541] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with Delta Rule. In *International Conference on Learning Representations*, 2025. 6, 9, 32, 33, 35, 37, 38, 39
- [542] Xiao-Wen Yang, Ziyu Han, Xi-Hua Zhang, Wen-Da Wei, Jie-Jing Shao, Lan-Zhe Guo, and Yu-Feng Li. Stabilizing Recurrent Dynamics for Test-Time Scalable Latent Reasoning in Looped Language Models. In *International Conference on Machine Learning*, 2026. 9, 66
- [543] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the Softmax Bottleneck: A High-Rank RNN Language Model. In *International Conference on Learning Representations*, 2018. 59
- [544] Da Yu, Edith Cohen, Badih Ghazi, Yangsibo Huang, Pritish Kamath, Ravi Kumar, Daogao Liu, and Chiyuan Zhang. Scaling Embedding Layers in Language Models. In *Neural Information Processing Systems*, 2025. 7, 9, 59
- [545] Fangyuan Yu. Scaling LLM Pre-Training with Vocabulary Curriculum. Preprint arXiv:2502.17910, 2025. 7, 9, 59
- [546] Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting Million-Byte Sequences with Multiscale Transformers. In *Neural Information Processing Systems*, 2024. 9, 58
- [547] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y X Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention. In *Association for Computational Linguistics*, 2025. 5, 9, 20, 23
- [548] Ted Zadouri, Hubert Strauss, and Tri Dao. Hardware-Efficient Attention for Fast Decoding. Preprint arXiv:2505.21487, 2025. 9, 28
- [549] Z.ai. GLM-4.7-Flash Model Card and Configuration. <https://huggingface.co/zai-org/GLM-4.7-Flash>, 2026. Official release; configuration at config.json. Accessed 2026-09-17. 69, 77
- [550] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. Preprint arXiv:1409.2329, 2014. 11
- [551] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. GLM-130B: An Open Bilingual Pre-Trained Model. In *International Conference on Learning Representations*, 2023. 73

- [552] Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, et al. PanGu- α : Large-Scale Autoregressive Pretrained Chinese Language Models with Auto-Parallel Computation. Preprint arXiv:2104.12369, 2021. 72
- [553] Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. AdaMoE: Token-Adaptive Routing with Null Experts for Mixture-of-Experts Language Models. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024. 9, 42
- [554] Aozhong Zhang, Selcuk Gurses, Yanxia Deng, Naigang Wang, Chi-Chun Liu, Davis Wertheimer, Derrick Liu, Xin Li, Zi Yang, Felix X.-F. Ye, and Penghang Yin. DropKV: Decoupling Residual-Output Perturbation for Near-Optimal KV-Cache Eviction. In *International Conference on Machine Learning*, 2026. URL <https://openreview.net/forum?id=MqfNzH3TVH>. 20, 21
- [555] Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Neural Information Processing Systems*, 2019. 7, 9, 48, 49
- [556] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. PQCache: Product Quantization-Based KVCache for Long Context LLM Inference. *Proceedings of the ACM on Management of Data*, 2025. 20, 22
- [557] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding. In *Association for Computational Linguistics*, 2024. 62
- [558] Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning Harmonized Representations for Speculative Sampling. In *International Conference on Learning Representations*, 2025. 62
- [559] Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Re. The Hedgehog & the Porcupine: Expressive Linear Attentions with Softmax Mimicry. In *International Conference on Learning Representations*, 2024. 34, 35
- [560] Michael Zhang, Aaryan Singhal, Benjamin Frederick Spector, Simran Arora, and Christopher Re. Low-Rank Linearization of Large Language Models. In *Workshop on Efficient Systems for Foundation Models II @ ICML*, 2024. 70, 75
- [561] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open Pre-Trained Transformer Language Models. Preprint arXiv:2205.01068, 2022. 67, 73
- [562] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-Level Convolutional Networks for Text Classification. In *Neural Information Processing Systems*, 2015. 57, 59
- [563] Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of Attention Heads: Selecting Attention Heads Per Token. In *Empirical Methods in Natural Language Processing*, 2022. 6, 9, 43
- [564] Xuan Zhang, Cunxiao Du, Chao Du, Tianyu Pang, Wei Gao, and Min Lin. SimLayerKV: A Simple Framework for Layer-Level KV Cache Reduction. *Transactions on Machine Learning Research*, 2025. 20, 21
- [565] Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John C. S. Lui, and Haibo Chen. Unifying KV Cache Compression for Large Language Models with LeanKV. Preprint arXiv:2412.03131, 2024. 20, 21
- [566] Yichi Zhang, Bofei Gao, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, Wen Xiao, et al. PyramidKV: Dynamic KV Cache Compression Based on Pyramidal Information Funneling. In *Conference on Language Modeling*, 2025. 9, 19, 20
- [567] Yifan Zhang, Yifeng Liu, Huizhuo Yuan, Zhen Qin, Yang Yuan, Quanquan Gu, and Andrew Chi-Chih Yao. Tensor Product Attention Is All You Need. In *Advances in Neural Information Processing Systems*, 2025. Spotlight; arXiv:2501.06425. 9, 28

- [568] Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, et al. CPM-2: Large-Scale Cost-Effective Pre-Trained Language Models. *AI Open*, 2021. 68, 72, 76
- [569] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer Feed-Forward Layers Are Mixtures of Experts. In *Findings of the Association for Computational Linguistics*, 2022. 41
- [570] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H₂O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *Neural Information Processing Systems*, 2023. 9, 19, 20
- [571] G. Zhao et al. Self-Distillation for Multi-Token Prediction. Preprint arXiv:2603.23911, 2026. 63
- [572] Junqi Zhao, Zhijin Fang, Shu Li, Shaohui Yang, and Shichao He. BUZZ: Beehive-Structured Sparse KV Cache with Segmented Heavy Hitters for Efficient LLM Inference. Preprint arXiv:2410.23079, 2024. 19, 20
- [573] Liang Zhao, Xiaocheng Feng, Xiachong Feng, Bin Qin, and Ting Liu. Length Extrapolation of Transformers: A Survey from the Perspective of Position Encoding. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024. 7
- [574] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A Survey of Large Language Models. *Frontiers of Computer Science*, 2026. 7
- [575] Chuanyang Zheng, Yihang Gao, Han Shi, Minbin Huang, Jingyao Li, Jing Xiong, Xiaozhe Ren, Michael Ng, Xin Jiang, Zhenguo Li, et al. DAPE: Data-Adaptive Positional Encoding for Length Extrapolation. In *Neural Information Processing Systems*, 2024. 9, 53, 54, 57
- [576] Chuanyang Zheng, Yihang Gao, Guoxuan Chen, Han Shi, Jing Xiong, Xiaozhe Ren, Chao Huang, Zhenguo Li, and Yu Li. Self-Adjust Softmax. In *Empirical Methods in Natural Language Processing*, 2025. 47
- [577] Shu Zhong, Mingyu Xu, Tenglong Ao, and Guang Shi. Understanding Transformer from the Perspective of Associative Memory. Preprint arXiv:2505.19488, 2025. 40
- [578] Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. DynamicKV: Task-Aware Adaptive KV Cache Compression for Long Context LLMs. In *Findings of the Association for Computational Linguistics: EMNLP*, 2025. 9, 19, 20
- [579] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-Experts with Expert Choice Routing. In *Neural Information Processing Systems*, 2022. 6, 9, 41, 42, 44
- [580] Yanqi Zhou, Nan Du, Yanping Huang, Daiyi Peng, Chang Lan, Da Huang, Siamak Shakeri, David So, Andrew M Dai, Yifeng Lu, et al. Brainformers: Trading Simplicity for Efficiency. In *International Conference on Machine Learning*, 2023. 45
- [581] Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving Speculative Decoding via Knowledge Distillation. In *International Conference on Learning Representations*, 2024. 9, 63
- [582] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A Survey on Efficient Inference for Large Language Models. Preprint arXiv:2404.14294, 2024. 7
- [583] Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-Connections. In *International Conference on Learning Representations*, 2025. 9, 49, 50, 51
- [584] Defa Zhu, Hongzhi Huang, Jundong Zhou, Zihao Huang, Yutao Zeng, Banggu Wu, Qiyang Min, and Xun Zhou. Frac-Connections: Fractional Extension of Hyper-Connections. Preprint arXiv:2503.14125, 2025. 49

- [585] Jiachen Zhu, Xinlei Chen, Kaiming He, Yann LeCun, and Zhuang Liu. Transformers without Normalization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 51
- [586] Qianchao Zhu, Jiangfei Duan, Chang Chen, Siran Liu, Xiuhong Li, Guanyu Feng, Xin Lv, Xiao Chuanfu, Dahua Lin, and Chao Yang. SampleAttention: Near-Lossless Acceleration of Long Context LLM Inference with Adaptive Structured Sparse Attention. In *Machine Learning and Systems*, 2025. 20, 22
- [587] Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling Latent Reasoning via Looped Language Models. Preprint arXiv:2510.25741, 2025. 7, 9, 66
- [588] Zhijian Zhuo, Ya Wang, Yutao Zeng, Xiaoqing Li, Xun Zhou, and Jinwen Ma. Polynomial Composition Activations: Unleashing the Dynamics of Large Language Models. In *International Conference on Learning Representations*, 2025. 41
- [589] Zhijian Zhuo, Yutao Zeng, Ya Wang, Sijun Zhang, Jian Yang, Xiaoqing Li, Xun Zhou, and Jinwen Ma. HybridNorm: Towards Stable and Efficient Transformer Training via Hybrid Normalization. In *Neural Information Processing Systems*, 2025. 9, 50
- [590] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing Stable and Transferable Sparse Expert Models. Preprint arXiv:2202.08906, 2022. 6, 9, 41, 42, 43
- [591] Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. MLKV: Multi-Layer Key-Value Heads for Memory Efficient Transformer Decoding. Preprint arXiv:2406.09297, 2024. 5, 9, 29
- [592] Jingwei Zuo, Maksim Velikanov, Dhia Eddine Rhaïem, Ilyas Chahed, Younes Belkada, Guillaume Kunsch, and Hakim Hacid. Falcon Mamba: The First Competitive Attention-Free 7B Language Model. Preprint arXiv:2410.05355, 2024. 68, 73
- [593] Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Jianfeng Gao, and Tuo Zhao. Taming Sparsely Activated Transformer with Stochastic Experts. In *International Conference on Learning Representations*, 2022. 44