ISSN: 2581-3404 (Online) **IF: 5.68 (SJIF)**

IJIRTM, Volume-8, Issue-5, October-2024.

Mixture-of-Experts Architectures: Sparse Activation, Routing Strategies, and Their Role in Scaling Large Language Models Efficiently

Kanish[1], Vineet Kumar[2], Harsh Sharma[3], Dipanshu[4], Sunita[5]

Department of Computer Science and Engineering (AI & ML) [1,2,3,4,5]

Panipat Institute of Engineering and Technology (PIET), Samalkha, Panipat, Haryana, India[1,2,3,4,5]

2822356.cse@pietgroup.co.in[1], 2822366.cse@pietgroup.co.in[2], 2822335.cse@pietgroup.co.in[3], 2822316.cse@pietgroup.co.in[4], sunita.cse-aiml@piet.co.in[5]

Abstract: *Mixture-of-Experts (MoE) has emerged as one of the most influential architectural paradigms for scaling large language models (LLMs) beyond the limits imposed by dense computation. By replacing a single monolithic feed-forward network with a bank of specialized sub-networks ("experts") and a lightweight learned router, MoE models decouple total parameter count from the computational cost of a forward pass, activating only a small, input-dependent subset of parameters for every token. This sparse activation principle allows contemporary MoE-based LLMs to reach hundreds of billions or even trillions of parameters while incurring inference and training costs comparable to much smaller dense models. This paper presents a comprehensive survey and technical analysis of MoE architectures, covering the historical evolution of the paradigm from early gating networks to modern trillion-parameter systems; the mathematical foundations of sparse activation and expert layer placement; a taxonomy of routing strategies including token-choice top-1 and top-k routing, expert-choice routing, hash-based routing, and auxiliary-loss-free load balancing; the systems-level challenges of memory, communication, and expert parallelism that determine real-world scaling efficiency; and detailed case studies of landmark models including GShard, Switch Transformer, GLaM, Mixtral, DeepSeek-MoE, and Grok-1. We further discuss training stability issues such as expert collapse and router oscillation, summarize open research challenges, and outline promising directions including fine-grained expert segmentation, heterogeneous expert capacity, and multimodal MoE. Our analysis indicates that routing strategy and load-balancing design are the dominant factors governing the practical efficiency gains that MoE architectures deliver, more so than raw expert count alone.*

Keywords: - Mixture-of-Experts, Sparse activation, Conditional computation, Routing strategies, top-k gating, Expert parallelism, Large language models, Model scaling, Load balancing, Transformer architecture.

Introduction

The past several years have witnessed an extraordinary escalation in the parameter counts of large language models (LLMs), driven by empirically observed scaling laws that relate model size, dataset size, and compute budget to downstream performance. Under the conventional dense transformer paradigm, however, every parameter in the model participates in the computation for every input token, so increasing capacity by increasing parameter count causes a proportional increase in the floating-point operations (FLOPs) required for both training

and inference. This coupling between capacity and compute has become a central bottleneck: organizations that wish to train ever-larger dense models face compute and energy costs that scale linearly, and eventually super-linearly once communication and memory-bandwidth constraints are taken into account.

Mixture-of-Experts (MoE) architectures offer a structurally different answer to this problem. Rather than routing every token through the entire network, an MoE layer partitions a dense feed-forward block into a collection of smaller expert sub-networks and introduces a learned router that selects, for each token, only a small subset of experts to activate. Because unselected experts contribute no computation for a given token, the total parameter count of the model can be increased dramatically — by adding more experts — without a corresponding increase in the compute cost of processing each token. This decoupling of parameter count from per-token compute is generally referred to as sparse activation or conditional computation, and it is the central mechanism by which MoE enables efficient scaling.

The idea of combining multiple specialized sub-networks under a gating mechanism predates the transformer era by decades, but it is only in the context of large-scale transformer-based language modeling that MoE has become a mainstream architectural choice for state-of-the-art systems. Contemporary production and open-weight models spanning research and industry — including systems commonly associated with names such as GShard, Switch Transformer, GLaM, Mixtral, DeepSeek-MoE, and Grok-1 — all rely on some variant of sparse expert routing to reach hundreds of billions to over a trillion total parameters while keeping the active, per-token compute in a far more modest range.

This paper aims to provide a self-contained technical treatment of MoE architectures suitable for readers already familiar with the standard transformer architecture. Section II reviews the historical background of MoE. Section III formalizes the architectural foundations of sparse expert layers. Section IV develops a taxonomy of routing strategies and load-balancing techniques. Section V analyzes the systems-level factors — compute, memory, and communication — that determine real-world scaling efficiency. Section VI presents case studies of prominent MoE-based models. Section VII discusses training stability challenges. Section VIII surveys applications of MoE beyond text-only language modeling. Section IX presents empirical scaling observations reported across the literature. Section X offers a comparative synthesis. Section XI discusses open challenges and future directions, and Section XII concludes the paper.

The contribution of this paper is not a new architecture or a new experimental result, but a structured synthesis: we organize a heterogeneous body of architectural, algorithmic, and systems work under a single consistent notation, we make explicit the quantitative relationships between sparsity ratio, expansion ratio, and communication cost that are often left implicit in individual system papers, and we juxtapose design choices across systems that are not always directly compared to one another in the original literature. Readers primarily interested in a specific sub-topic — for instance, load-balancing algorithms alone — may proceed directly to the corresponding subsection without loss of continuity, since each major section is written to be reasonably self-contained.

Background and Evolution of Mixture-of-Experts

The conceptual origin of Mixture-of-Experts dates to early work on adaptive mixtures of local experts, which proposed training multiple neural networks jointly with a gating network that learns to divide the input space among them so that each expert specializes in a distinct region of the problem. This early formulation already contained the two ingredients that define modern MoE: a set of expert networks and a trainable gate that combines their outputs, typically through a weighted sum.

The modern, large-scale incarnation of this idea for deep learning began with the introduction of a sparsely-gated mixture-of-experts layer inserted between stacked recurrent or feed-forward layers, in which a noisy top-k gating function was used to activate only a handful of experts out of a much larger pool, and where explicit load-balancing losses were introduced to prevent the router from collapsing onto a small number of favored experts. This work demonstrated that a mixture layer with thousands of experts could be trained with a very large increase in capacity at only a modest increase in computation, and it established the practical template — noisy top-k routing plus an auxiliary balancing loss — that nearly all subsequent MoE transformer work builds upon.

The transition of MoE into the transformer era occurred when researchers replaced the feed-forward sub-layer of alternating transformer blocks with a sparsely-gated expert layer and combined this with automatic operation-level sharding across large accelerator pods, scaling multilingual translation models to hundreds of billions of parameters. This showed that MoE layers could be combined with model-parallel and expert-parallel training strategies to reach scales that would be impractical for dense architectures on the same hardware budget.

A subsequent line of work simplified the routing decision to a single top-1 expert per token, reducing router complexity, communication volume, and computational overhead while introducing techniques for numerical stability such as selective precision and improved initialization, enabling stable training of models reported to reach on the order of a trillion parameters. In parallel, other efforts explored alternating dense and sparse layers with the explicit goal of maximizing sample and energy efficiency rather than raw parameter count, arguing that a well-tuned MoE model could match the quality of a much larger dense model while consuming substantially less energy over training.

More recent open-weight releases have popularized top-2 token-choice routing over a modest number of experts (commonly eight), demonstrating that a sparse model with roughly one-quarter of its parameters active per token can match or exceed the quality of dense models several times larger while offering substantially faster inference at low batch sizes. The most recent generation of large MoE systems has pushed toward fine-grained expert segmentation, in which each feed-forward block is decomposed into a much larger number of smaller experts with several activated per token, together with one or more always-active “shared” experts that capture common knowledge, and with load-balancing mechanisms that avoid the accuracy cost traditionally imposed by auxiliary balancing losses.

Beyond individual model releases, a growing body of survey and systems literature has attempted to organize the resulting design space into coherent taxonomies, distinguishing architectural questions (how many experts, where to place them, how large each expert should be) from algorithmic questions (how tokens are matched to experts and how load is balanced) and from systems questions (how experts are sharded across accelerators and how communication is minimized). This three-way decomposition — architecture, algorithm, and systems — is a useful lens for the remainder of this paper, since a given empirical outcome (for example, a reported inference speedup) is often attributable to a specific combination of choices along all three axes rather than to “MoE” as a single monolithic technique.

It is also worth noting that MoE has, at different points, been motivated by different primary objectives. Some efforts have emphasized raw capacity — how large a model can be trained at all within a fixed compute budget — while others have emphasized energy or sample efficiency at a fixed quality target, and still others have emphasized inference latency and serving cost for a fixed level of downstream capability. Because these objectives can favor different points in the design space (for instance, a higher active-parameter fraction may improve quality per training step but worsen serving cost), it is important to interpret comparisons between MoE systems, including those summarized later in this paper, with attention to which objective each system was primarily optimized for.

It is worth situating MoE relative to two related but distinct ideas with which it is sometimes conflated. Ensemble methods also combine multiple sub-models, but conventional ensembles evaluate every member model for every input and combine their outputs at inference time, incurring compute proportional to the number of members; MoE differs precisely because only a small, input-dependent subset of experts is ever evaluated, making it a form of conditional computation rather than an ensembling technique. Conditional computation more broadly encompasses any mechanism, including early-exit networks and dynamic-depth architectures, in which the amount or path of computation applied to an input varies based on the input itself; MoE can be understood as one particular, and currently the most empirically successful, instance of this broader family applied to the width (rather than the depth) of a network's feed-forward computation.

Architectural Foundations of Sparse Mixture-of-Experts

A. The Sparse Activation Principle

In a standard dense transformer, each token representation x is passed through a feed-forward network (FFN) consisting of two linear transformations with a nonlinearity in between. In an MoE transformer, this single FFN is replaced by a set of N expert networks E1, E2, …, EN, each structurally identical to a standard FFN (or, in some designs, deliberately smaller), together with a router (or gating network) G that maps the token representation to a probability distribution over the N experts. The output of the MoE layer for token x is a weighted combination of the outputs of the selected experts:

$$y(x) = \Sigma_{i \in \text{Top-k}(G(x))} G(x)_i \cdot E_i(x)$$

where G(x) is typically computed by a single learned linear projection of x followed by a softmax, and Top-k(·) denotes the indices of the k experts with the largest gating values. Crucially, experts whose gate value is not among the top-k are assigned a weight of exactly zero and are never evaluated for that token, which is what makes the computation sparse: the FLOPs required to process a token depend only on k and the size of an individual expert, not on N, the total number of experts. This is the essential mechanism that decouples model capacity (governed by N) from per-token compute (governed by k).

B. Placement of Expert Layers

MoE layers are not typically used to replace every component of the transformer; the self-attention sub-layers are almost always left dense, since attention is comparatively cheap relative to the FFN and specializing attention heads by input is far less well understood than specializing feed-forward computation. Design choices differ mainly in how frequently the FFN sub-layer is replaced by an MoE layer. Some architectures replace every FFN sub-layer with an MoE layer to maximize the fraction of parameters that are sparse; others alternate dense and sparse FFN layers, interleaving a standard dense block after every MoE block, which has been argued to stabilize training and to preserve some capacity that is guaranteed to process every token identically, improving certain forms of generalization and reducing the router's burden.

C. The Router / Gating Network

The router is architecturally simple — usually nothing more than a single dense layer mapping the model's hidden dimension d to the number of experts N, followed by a softmax — yet it is functionally the most consequential component of the entire MoE layer, since it determines both which experts specialize in which kinds of tokens and how evenly compute is spread across the available hardware. Because the router's argmax or top-k operation is not differentiable in the traditional sense, gradients flow to the router only through the softmax weighting of the selected experts' outputs; experts that are never selected for a given token receive no gradient signal from that token at all. This creates a fundamental tension, discussed further in Section VII, between encouraging the router

to specialize (so that experts learn distinct, useful functions) and encouraging it to balance load evenly across experts (so that hardware utilization remains high and no expert is starved of training signal).

D. Quantifying Sparsity, Capacity, and Load

It is useful to define a small set of quantities that recur throughout the remainder of this paper. The sparsity ratio of an MoE layer is defined as k / N, the fraction of experts activated per token; smaller values correspond to sparser, more efficient layers for a fixed N. The expansion ratio is defined as N_total / N_active, the factor by which total parameters exceed active parameters; this quantity is often reported directly by system developers as a headline efficiency statistic. The per-expert load for expert i over a batch of T tokens is the count of tokens for which expert i appears among the top-k selections; under perfectly balanced routing, every expert would receive an expected load of kT / N tokens. Finally, the capacity factor c determines the maximum load an expert will actually process, typically set as $c \cdot (kT / N)$ for some $c \geq 1$, with any excess tokens dropped. These four quantities — sparsity ratio, expansion ratio, per-expert load, and capacity factor — jointly determine the compute, memory, and quality profile of a given MoE configuration, and much of the design work described in Sections IV and V can be understood as different strategies for controlling the relationship between them.

E. Internal Expert Architecture Variants

Most descriptions of MoE treat each expert as an unspecified black-box FFN, but the internal architecture chosen for each expert has a measurable effect on both quality and efficiency. The majority of contemporary systems use SwiGLU-style gated feed-forward blocks as the expert function, following the broader trend in dense transformer design, rather than the simpler ReLU-based FFN used in the original sparsely-gated MoE and early Switch Transformer work. Expert size is another design axis: some systems use experts identical in width to what a dense FFN of the same model would use, while fine-grained designs deliberately shrink each individual expert's hidden dimension so that more experts can be trained within the same total parameter budget, trading per-expert representational capacity for a larger and more combinatorially rich pool of specialists. A smaller number of systems have also experimented with heterogeneous expert sizes within the same layer, though this remains uncommon in production systems due to the added complexity it introduces for capacity planning and load balancing.

Routing Strategies

The routing strategy — the algorithm by which tokens and experts are matched — is the primary axis along which MoE architectures differ from one another, and it has a direct and substantial impact on model quality, training stability, and system efficiency. Table II summarizes the principal families of routing strategies used in current systems; each is discussed in more detail below.

A. Token-Choice Top-1 Routing

In top-1 (or “switch”) routing, each token is assigned to exactly one expert: the single expert with the highest router probability. This is the simplest and cheapest routing strategy, since it minimizes both the FLOPs used per token (a single expert forward pass) and the volume of data that must be exchanged between accelerators when experts are distributed across devices, because each token needs to be sent to only one destination. The principal drawback of top-1 routing is that it offers the router no redundancy: if the router makes a poor assignment for a given token, there is no second expert to partially compensate, and the training signal for the router itself is sparser than in top-k schemes with k>1.

B. Token-Choice Top-k Routing (k > 1)

Top-k routing with k=2 or higher, popularized by large-scale multilingual MoE systems and later adopted by several widely used open-weight models, routes each token to its k highest-scoring experts and combines their

outputs as a weighted sum using the (renormalized) router probabilities. Empirically, top-2 routing tends to produce higher-quality models than top-1 for a comparable active-parameter budget, since the token representation benefits from the combined perspective of two specialized sub-networks and the router receives a denser training signal. The cost is a roughly proportional increase in per-token FLOPs and, more significantly, in inter-device communication volume when experts are sharded across accelerators, since each token must now be routed to (and its result gathered from) two separate devices rather than one.

C. Expert-Choice Routing

Token-choice routing schemes let each token select its preferred experts, which can lead to severe load imbalance: popular experts may be selected far more often than the average, forcing the system either to drop excess tokens or to pad under-utilized experts with wasted compute. Expert-choice routing inverts this relationship: instead of tokens choosing experts, each expert selects a fixed-size buffer of the tokens it scores most highly from the current batch. Because every expert fills exactly the same buffer size, load is balanced by construction, which removes the need for a separate load-balancing auxiliary loss. The trade-off is that a given token may be selected by zero experts (and thus dropped from sparse processing for that layer) or by more experts than intended, and expert-choice routing requires access to a batch of tokens at routing time, which is a more natural fit for training and batched inference than for strictly causal, one-token-at-a-time autoregressive generation.

D. Auxiliary Load-Balancing Losses

To discourage token-choice routers from collapsing onto a small subset of experts, most token-choice systems add an auxiliary loss term to the overall training objective that penalizes uneven expert utilization across a batch. A common formulation multiplies, for each expert, the fraction of tokens routed to that expert by the average router probability assigned to that expert across the batch, and sums this product over all experts; minimizing this quantity jointly with the task loss pushes both the discrete routing decisions and the continuous router probabilities toward a uniform distribution over experts. A second auxiliary term, sometimes called a router z-loss, penalizes large router logits directly, which improves numerical stability, particularly when training is carried out in reduced precision.

E. Auxiliary-Loss-Free Load Balancing

Auxiliary balancing losses are effective at equalizing load, but because they are added directly to the training objective, they necessarily compete with the primary language-modeling loss, and an overly large balancing coefficient can measurably harm model quality. Recent large-scale systems have introduced a load-balancing mechanism that instead maintains a per-expert bias term, added to the router's logits purely for the purpose of the routing decision (and not included in the gradient path for the task loss), which is incremented for experts that are currently under-utilized and decremented for experts that are over-utilized, based on observed load statistics after each training step. This decouples load balancing from the gradient-based training signal entirely, allowing near-uniform expert utilization to be achieved without the quality trade-off historically associated with auxiliary losses.

F. Hash-Based and Fixed Routing

A simpler alternative dispenses with a learned router altogether, assigning tokens to experts via a deterministic hash function of the token identity or position. Hash-based routing guarantees a perfectly balanced load by construction and removes router parameters and their associated instability, at the cost of forfeiting any learned, content-based specialization; in practice, hash routing has mainly served as a useful ablation baseline that helps isolate how much of MoE's benefit comes from adaptive routing versus sparse computation alone, and it is largely deprecated in current large-scale systems.

G. Soft and Differentiable Routing Variants

A separate line of work has explored soft MoE formulations in which every expert processes a weighted combination of tokens (or every token is represented as a weighted combination of a fixed number of learned slots) rather than a hard, discrete subset, entirely avoiding the non-differentiable top-k operation and its associated load-balancing machinery. Soft routing removes token dropping and load imbalance by construction, since every expert always receives the same, fixed-size input regardless of the underlying data distribution, but it does so by relaxing the hard sparsity that is essential to the FLOPs savings described in Section V; some soft variants are therefore better understood as a regularization or specialization technique for moderate-sized expert pools rather than as a direct route to trillion-parameter-scale sparse computation. Hard, discrete top-k or expert-choice routing consequently remains the dominant paradigm for the largest-scale production and open-weight systems discussed in Section VI.

H. Choosing a Routing Strategy in Practice

Given the taxonomy above, practitioners selecting a routing strategy for a new system face a multi-way trade-off rather than a single dominant choice. Top-1 routing remains attractive when minimizing communication and implementation complexity is paramount, such as when training must span many loosely coupled nodes. Top-2 or top-k routing is generally preferred when a moderate increase in communication and compute is acceptable in exchange for improved quality, and it remains the most common choice among openly released models of moderate scale. Expert-choice routing is attractive whenever training-time batched access to tokens is available and near-perfect load balance is a priority, but its incompatibility with strictly sequential autoregressive decoding limits its direct use at inference time. Auxiliary-loss-free balancing is increasingly favored for the largest-scale systems precisely because it removes the tension between load balancing and task quality that motivated much of the load-balancing research summarized in this section in the first place.

Table 1: COMPARISON OF ROUTING STRATEGIES IN SPARSE MoE ARCHITECTURES.

Strategy	Selection Unit	Key Mechanism	Primary Advantage	Primary Drawback
Token-Choice Top-1	Token picks 1 expert	Argmax over router logits	Minimal compute & communication	Higher risk of load imbalance
Token-Choice Top-k (k>1)	Token picks k experts	Weighted sum of top-k expert outputs	Richer representation, better quality	Higher FLOPs and all-to-all traffic
Expert-Choice Routing	Expert selects tokens	Each expert picks its top buffer of tokens	Near-perfect load balance by design	Tokens may be dropped or duplicated
Auxiliary-Loss-Free Balancing	Token picks experts; bias adjusted post-hoc	Per-expert bias term updated by observed load	Balances load without harming task loss	Requires careful bias update scheduling
Hash / Fixed Routing	Token mapped deterministically	Hash function or fixed assignment	No router parameters, fully balanced	No learned specialization signal

Scaling Efficiency: Compute, Memory, and Communication

A. Decoupling FLOPs from Parameter Count

The central efficiency claim of MoE architectures is that total parameter count N_total and active parameter count N_active can be scaled almost independently. For a token-choice top-k MoE layer with N experts of equal size, N_total grows linearly with N, while the FLOPs required to process one token depend only on k and the per-expert size, both of which can be held fixed as N grows. This means a practitioner can increase model capacity — and, empirically, downstream quality — by adding experts, while the compute budget for training and inference grows far more slowly than it would for a dense model reaching the same total parameter count. This is why MoE models with hundreds of billions of total parameters can be trained and served at a compute cost comparable to dense models an order of magnitude smaller in active parameters.

B. Memory Footprint and Expert Parallelism

While MoE decouples compute from parameter count, it does not decouple memory footprint from parameter count: every expert's weights must reside somewhere in accelerator memory (or be paged in from host memory or storage) regardless of whether that expert is activated for a given token, because different tokens in the same batch typically activate different experts. This makes memory capacity, not raw FLOPs, the binding constraint for very large MoE models, and it motivates expert parallelism, a distribution strategy in which different experts are physically hosted on different accelerators or accelerator groups, distinct from the more familiar data, tensor, and pipeline parallelism strategies used for dense models. In practice, large MoE systems combine expert parallelism with these other strategies, sharding experts across nodes while replicating or tensor-splitting the dense components (attention, embeddings, router) of the network.

C. All-to-All Communication Overhead

Expert parallelism introduces a systems cost that dense models do not incur: after the router makes its assignment, tokens must physically be sent across the interconnect to the device hosting their selected expert(s), processed, and then the results must be sent back to be recombined with the rest of the sequence. This is typically implemented as an all-to-all collective communication operation, executed twice per MoE layer (dispatch and combine). The volume of this traffic scales with k (the number of experts activated per token) and with the degree of expert parallelism, and because all-to-all communication is latency-sensitive and depends on interconnect bandwidth, it can become the dominant cost of an MoE forward or backward pass at large scale, particularly across multiple physical nodes rather than within a single high-bandwidth accelerator pod. This is one of the principal reasons top-1 routing remains attractive for the largest-scale systems despite its quality trade-off relative to top-2 or top-k routing, and why fine-grained experts are typically combined with efficient sharding layouts that minimize cross-node token traffic.

D. Capacity Factor and Token Dropping

Because expert assignment is data-dependent, the number of tokens routed to a given expert within a batch is a random variable, not a fixed quantity, yet accelerator kernels generally require statically-shaped buffers for efficient execution. Systems address this by defining a per-expert capacity — the maximum number of tokens an expert will process in a given batch, usually expressed as a capacity factor multiplying the batch size divided evenly by the number of experts. Tokens routed to an expert that has already reached capacity are dropped from sparse computation at that layer, typically passed forward via a residual connection instead. A larger capacity factor reduces token dropping and improves quality but increases wasted compute from padding under-filled experts, so the capacity factor is an important and somewhat delicate hyperparameter governing the trade-off between quality and efficiency.

E. Fine-Grained Expert Segmentation

An important recent refinement is fine-grained expert segmentation, in which a fixed FFN parameter budget is divided into a much larger number of smaller experts (for example, splitting what would have been eight large experts into sixty-four smaller ones) while proportionally increasing the number of experts activated per token, so that active parameter count is held constant. This increases the number of distinct expert combinations available to the router combinatorially, which has been shown empirically to improve specialization and downstream quality at fixed active-parameter and total-parameter budgets. Several recent large-scale systems additionally designate one or more experts as always-active "shared" experts, which are applied to every token regardless of the router's decision, intended to absorb common, cross-domain knowledge so that the remaining routed experts can specialize more narrowly.

F. A Worked Numeric Illustration

Consider a hypothetical MoE layer replacing a dense FFN of 1 billion parameters. If this layer is instead built from N = 16 experts of 1 billion parameters each, with top-k = 2 routing, the total parameter count of the layer grows sixteen-fold to 16 billion, while the per-token compute grows only two-fold, since exactly two experts are evaluated per token regardless of N. The expansion ratio, as defined in Section III-D, is 16 / 2 = 8, meaning the layer offers eight times more capacity per unit of per-token compute than the original dense layer. Extending this reasoning across every FFN layer in a deep transformer, and combined with the fact that attention layers remain dense and therefore do not benefit from this expansion, illustrates why whole-model expansion ratios reported for real systems (see Table I) are typically somewhat smaller than the per-layer expert-count ratio alone would suggest, since a full accounting must weight both dense and sparse components by their respective share of total FLOPs.

G. Interplay with Tensor and Pipeline Parallelism

In practice, expert parallelism is rarely used in isolation; large MoE systems typically combine it with tensor parallelism (splitting individual weight matrices, including the dense attention and embedding components, across accelerators within a high-bandwidth interconnect group) and pipeline parallelism (splitting the network's layers across sequential stages, each hosted on a different group of accelerators). A common configuration places expert-parallel groups within a single high-bandwidth node or tightly coupled node group, where all-to-all communication is cheapest, while using pipeline parallelism to span across more loosely coupled groups of nodes, since pipeline communication (activations passed between adjacent stages) is inherently more tolerant of lower bandwidth and higher latency than the all-to-all pattern required by expert parallelism. Choosing this multi-dimensional parallelism layout correctly is widely regarded as one of the more difficult practical aspects of training frontier-scale MoE models, and it is often as consequential for achieved training throughput as the choice of routing algorithm itself.

Case Studies of Prominent Moe Models

This section reviews several landmark systems that illustrate the design space described above; a condensed comparison appears in Table I.

A. GShard

GShard combined a top-2 token-choice MoE layer, inserted in every other transformer block, with automatic operation-level sharding annotations that allowed a compiler to distribute both dense and sparse components efficiently across thousands of accelerators. Applied to a multilingual machine translation model reaching several hundred billion parameters, GShard demonstrated that such models could be trained in a practical amount of wall-

clock time while substantially outperforming much smaller dense baselines on many-language translation, establishing MoE combined with automatic sharding as a viable path to very large model capacity.

From a systems perspective, GShard's contribution was as much about the sharding compiler as about the MoE layer itself: annotating tensors with placement hints allowed the same model definition to be automatically partitioned across thousands of accelerators without hand-written parallelization code for each configuration, a pattern that strongly influenced later expert-parallel training frameworks.

B. Switch Transformer

Switch Transformer simplified the routing decision to top-1 ("switch") routing, arguing that the reduced computational and communication cost, combined with careful initialization and selective use of higher-precision arithmetic in the router, more than compensated for the loss of the redundancy that top-2 routing provides. This simplification enabled training of sparse models reported to reach on the order of a trillion parameters while maintaining favorable sample efficiency relative to dense baselines of comparable active-parameter count, and it popularized the auxiliary load-balancing loss formulation that remains in wide use.

Switch Transformer's ablations were also notable for showing that, at matched active-parameter count and matched training FLOPs, sparse top-1 models reached a given training loss in substantially fewer training steps than their dense counterparts, providing some of the clearest early quantitative evidence that sparsity itself — not merely additional parameters — was responsible for the observed efficiency gains.

C. GLaM

GLaM used top-2 token-choice routing but alternated MoE layers with standard dense layers, and it emphasized energy efficiency as a primary evaluation metric rather than raw parameter count alone. Despite activating only a small fraction of its total parameters per token, GLaM was reported to match or exceed the quality of a substantially larger dense contemporary model while consuming markedly less energy to train, reinforcing the argument that sparse activation is fundamentally about compute and energy efficiency rather than parameter count as an end in itself.

The alternating dense/MoE design of GLaM can be viewed as a deliberate hedge: dense layers guarantee a baseline of uniform, well-understood computation for every token, while interleaved MoE layers supply the bulk of the additional capacity, an arrangement that later influenced the shared-expert designs discussed in Section III-B and Section VI-E.

D. Mixtral 8x7B

Mixtral popularized top-2 token-choice MoE at a much more accessible scale for the open-weight community, replacing every feed-forward block with a mixture of eight experts of standard transformer FFN structure, for a total of roughly 47 billion parameters of which around 13 billion are active for any given token. Despite its comparatively modest active-parameter count, Mixtral was reported to match or outperform a considerably larger dense model on most benchmarks, while offering substantially faster inference, particularly at small batch sizes, illustrating the practical latency benefits of sparse activation outside of hyperscale training infrastructure.

Mixtral's release was also significant for the open-source ecosystem specifically, since its relatively small number of coarse-grained experts (eight, each a full-sized FFN) made the model tractable to run and fine-tune on widely available multi-GPU workstations, in contrast to the thousands-of-experts configurations used in some earlier hyperscale systems, helping to popularize MoE as a practical architecture choice well beyond the largest industrial labs.

E. DeepSeek-MoE and DeepSeek-V3

The DeepSeek family of models introduced fine-grained expert segmentation together with one or more shared experts applied to every token, and its largest model combines this design with the auxiliary-loss-free load-

balancing mechanism described in Section IV-E. With a reported total parameter count in the neighborhood of several hundred billion, only a small fraction — on the order of tens of billions — is active per token, illustrating an aggressively sparse operating point relative to earlier systems and demonstrating that competitive quality with leading dense and sparse models alike can be achieved at a comparatively low active-parameter budget.

The combination of fine-grained experts with a small number of always-active shared experts is presented by its designers as a way of separating universally useful computation, handled by the shared experts, from more narrowly specialized computation, handled by the routed experts, and the accompanying auxiliary-loss-free balancing mechanism (Section IV-E) is reported to close much of the quality gap that auxiliary-loss-based balancing had historically introduced relative to an idealized, unconstrained router.

F. Grok-1

Grok-1 is an openly released MoE model using top-2 token-choice routing over eight experts, with several hundred billion total parameters and a substantially smaller active-parameter count per token, following broadly the same architectural template as Mixtral but at a larger overall scale, and it further illustrates that the top-2, eight-expert design point has become a common convention among independently developed open-weight MoE systems.

Table 2: COMPARATIVE OVERVIEW OF PROMINENT MoE-BASED LANGUAGE MODELS

Model	Year	Total Params	Active Params	Experts / Layer	Routing	Placement
GShard (MoE)	2020	~600B	~n/a (top-2 subset)	2048 (task-dependent)	Top-2 token-choice	Every other FFN
Switch Transformer	2021	up to 1.6T	~model-dependent	up to 2048	Top-1 (switch)	Every FFN layer
GLaM	2021	1.2T	~96B (~8%)	64	Top-2 token-choice	Alternating dense/MoE
Mixtral 8x7B	2023	46.7B	12.9B	8	Top-2 token-choice	Every FFN block
DeepSeek-MoE / V3	2024	up to 671B	~37B	fine-grained (many small)	Top-k + shared experts, aux-loss-free balancing	Every FFN layer
Grok-1	2024	314B	~86B	8	Top-2 token-choice	Every FFN block

Training Stability and Practical Challenges

A. Expert Collapse

Without adequate load-balancing pressure, MoE routers exhibit a well-documented tendency toward expert collapse, in which the router increasingly concentrates its probability mass on a small subset of experts early in training. Because those favored experts then receive a disproportionate share of the gradient signal, they continue to improve relative to the neglected experts, reinforcing the router's preference for them in a self-amplifying feedback loop. Left unchecked, this can reduce an MoE layer with many experts to an effectively much smaller model in practice, eliminating most of the intended capacity benefit. Auxiliary balancing losses, expert-choice

routing, and auxiliary-loss-free bias adjustment are all, from this perspective, different mechanisms for interrupting this feedback loop.

B. Router Numerical Instability

Because the router's softmax output directly scales the contribution of each expert, large or rapidly growing router logits can produce numerical instability, particularly when the surrounding network is trained in reduced-precision formats such as bfloat16. This has motivated techniques such as computing the router in higher precision even when the rest of the network runs in lower precision, and adding a router z-loss term that directly penalizes the magnitude of the logits feeding the softmax, both of which have been reported to substantially improve training stability at large scale with negligible additional cost.

C. Fine-Tuning and Sparse Overfitting

Sparse MoE models have also been observed to be more prone to overfitting during fine-tuning on comparatively small downstream datasets than dense models of similar active-parameter count, plausibly because individual experts, seeing a smaller effective slice of the fine-tuning data due to routing, can memorize that slice more readily. Mitigations reported in the literature include selectively freezing or applying stronger regularization to expert parameters while fine-tuning the router and other dense components more freely, and increasing the effective batch size seen by each expert during fine-tuning.

D. Inference-Time Considerations

Although MoE reduces the FLOPs required per token, it does not reduce the memory required to hold the full parameter set, which must typically remain resident (across one or more devices) to serve arbitrary requests whose routing pattern cannot be predicted in advance. This creates a distinctive inference profile: MoE models can be compute-efficient yet memory-hungry, favoring deployment configurations with abundant aggregate accelerator memory even when the compute demand per request is modest, and motivating techniques such as expert offloading, caching of frequently activated experts, and quantization of expert weights to reduce the effective memory footprint at serving time.

E. Monitoring and Diagnostics During Training

Because the pathologies described above — expert collapse, router logit blow-up, and uneven load — can develop gradually over many thousands of training steps before becoming catastrophic, practical MoE training pipelines typically track a small set of routing diagnostics throughout training: the distribution of tokens routed to each expert, the fraction of tokens dropped due to capacity limits, the entropy of the router's output distribution, and the magnitude of router logits. Sudden changes in any of these statistics are commonly used as an early warning signal that intervention (adjusting the balancing coefficient, the capacity factor, or the learning rate) may be needed before the underlying task loss visibly degrades, since routing pathologies often precede measurable loss degradation by a noticeable margin.

F. An Illustrative Debugging Scenario

To make the diagnostics of Section VII-E concrete, consider a representative (illustrative, not system-specific) scenario reported in various forms across the practitioner literature: partway through training, the task loss plateaus earlier than the learning-rate schedule would predict, while overall gradient norms remain unremarkable. Inspection of per-expert load statistics reveals that a small subset of experts is receiving a disproportionately growing share of tokens, while several others are receiving load near zero and are effectively no longer being trained. In this scenario, increasing the load-balancing coefficient (for auxiliary-loss-based systems) or the bias update rate (for auxiliary-loss-free systems) for a short window of training steps is often sufficient to redistribute load and allow the previously neglected experts to resume receiving useful gradient signal, after which the coefficient can typically be returned to its original value without re-triggering collapse. This kind of scenario illustrates why routing diagnostics are generally treated as a first-class, continuously monitored training signal in large-scale MoE pipelines rather than as a one-time configuration check performed only at the start of training.

Applications beyond Text-Only Language Modeling

Although this paper focuses on MoE in the context of text-based large language models, the same sparse activation principle has been applied successfully in adjacent domains, and reviewing these applications helps to distinguish which aspects of MoE are specific to language modeling and which are general properties of sparse conditional computation.

A. Computer Vision

Vision transformers have incorporated sparsely-gated MoE layers, typically replacing the FFN sub-layer in a manner directly analogous to text transformers, to scale image classification and detection models with sub-linear compute growth relative to parameter count. Because image patches are considerably more homogeneous in structure than text tokens, some vision-MoE work has reported needing larger capacity factors or specialized auxiliary losses to prevent experts from collapsing onto trivial, low-level distinctions such as average patch brightness rather than more semantically meaningful groupings.

B. Multimodal and Cross-Modal Systems

Multimodal systems have used mixtures of experts to allow different experts to specialize implicitly by modality or by task, sharing a common transformer backbone while routing image, text, audio, or other modality tokens through partially overlapping sets of experts. In several reported designs, router statistics reveal a natural, emergent tendency for certain experts to specialize predominantly in one modality even without explicit modality-conditioned routing, suggesting that content-based routing alone can discover structure that would otherwise require hand-designed, modality-specific sub-networks.

C. Speech and Multilingual Translation

Speech and multilingual translation systems, following the lineage of the original GShard work described in Section II, continue to use MoE to scale capacity across very large numbers of languages without a proportional increase in per-utterance compute. In this setting, experts have been observed to specialize partly along language-family lines, which is consistent with the broader observation that MoE routing tends to discover clusters that align with meaningful structure in the underlying data distribution whenever such structure exists.

D. Scientific and Structured-Data Applications

More recently, MoE has begun to appear in scientific machine learning contexts, including models over molecular graphs, tabular data, and time series, where heterogeneous experts can specialize by data regime (for example, by molecule size or by sequence length) rather than by language or modality. While this application area is comparatively less mature than text or vision MoE, the underlying motivation — heterogeneous data distributions benefiting from specialized sub-networks activated conditionally — is the same general principle discussed throughout this paper, suggesting that MoE is likely to remain relevant well beyond its origins in language modeling.

Empirical Scaling Behavior and Benchmarking Considerations

A. Scaling with Expert Count at Fixed Active Parameters

A recurring empirical finding across the systems surveyed in Section VI is that, for a fixed active-parameter budget and a fixed training compute budget, increasing the total number of experts (and hence the expansion ratio) tends to improve downstream quality with diminishing but still positive returns over a wide range, before eventually plateauing once the router's ability to make effective use of additional specialization is exhausted. This has been used to argue that, for a fixed inference cost, adding experts is frequently a more favorable use of additional training compute than increasing active parameters directly, provided that load-balancing and communication costs (Section V) remain manageable at the resulting expansion ratio.

B. Sensitivity to Batch Size and Sequence Length

Because routing decisions and capacity constraints operate over a batch of tokens, MoE model quality and hardware utilization can be more sensitive to batch size and sequence length than dense models are. Very small batches provide the router with a limited pool of tokens over which to balance load, increasing the relative impact of token dropping, while very large batches amortize load-balancing overhead more effectively but increase memory pressure from activations. This sensitivity is one of the reasons that reported comparisons between MoE and dense models are sometimes difficult to reproduce exactly across different training and serving configurations, and it underscores the importance of reporting batch size, sequence length, and capacity factor alongside any efficiency claim.

C. Benchmarking Caveats

Finally, when comparing reported benchmark numbers across the systems in Table I, it is important to note that these systems differ not only in MoE-specific design choices but also in training data composition, total training tokens, tokenizer, and evaluation protocol, any of which can materially affect benchmark scores independently of the MoE architecture itself. Claims that a given routing strategy is categorically "better" than another are therefore best interpreted as holding under the specific matched-compute or matched-active-parameter conditions reported in the corresponding original study, rather than as universal, condition-independent statements.

Comparative Synthesis

Table II summarizes the routing strategies discussed in Section IV alongside their principal trade-offs. Two overarching patterns emerge from comparing the systems in Table I and the strategies in Table II. First, the ratio of total to active parameters has grown substantially over time, from roughly single-digit multiples in early systems to several multiples or more in the most fine-grained recent designs, reflecting increasing confidence that very sparse activation ratios remain trainable given adequate load-balancing techniques. Second, the field has broadly converged away from pure auxiliary-loss-based balancing and toward either expert-choice routing or auxiliary-loss-free bias-based balancing, both of which achieve load balance without directly penalizing the primary task objective, suggesting that decoupling load balance from the gradient path is an important general principle rather than an implementation detail specific to any one system.

A third pattern, less immediately visible from Table I alone, concerns expert granularity. Early large-scale systems tended to use either a very large number of experts with relatively simple top-1 or top-2 routing (favoring communication efficiency) or a moderate number of comparatively large experts (favoring implementation simplicity), whereas the most recent systems increasingly favor many small, fine-grained experts combined with one or more always-active shared experts. This shift suggests that the field's understanding of the trade-off between router expressiveness and communication cost has matured: fine-grained experts increase the combinatorial richness of possible expert subsets without proportionally increasing communication volume, provided the routing and sharding implementation is designed to keep the corresponding all-to-all traffic manageable, which is itself an active systems engineering problem rather than a solved one.

Open Challenges and Future Directions

Despite the substantial progress summarized in the preceding sections, several aspects of MoE design remain open research problems rather than settled engineering practice. We highlight six such directions below, spanning architecture, algorithms, and systems, each of which represents a plausible source of further efficiency or quality gains beyond what current-generation systems achieve.

- Heterogeneous expert capacity: most current systems use experts of identical size and architecture; allowing experts to vary in capacity according to the complexity of the sub-tasks they specialize in remains comparatively underexplored.

• Dynamic and input-adaptive k: the number of experts activated per token is typically fixed in advance; allowing the model to activate more experts for difficult tokens and fewer for easy ones could improve the compute–quality trade-off.
• Communication-aware routing: incorporating the physical topology of the accelerator interconnect into the routing decision itself, rather than treating placement and routing as separate concerns, could reduce all-to-all overhead at very large scale.
• Robust interpretability of expert specialization: understanding what individual experts actually learn to specialize in remains largely empirical and post-hoc, limiting the ability to design routing objectives around desired specialization patterns directly.
• Efficient serving under memory constraints: techniques for expert offloading, caching, and quantization are still maturing relative to the corresponding techniques available for dense models, and remain an active systems research area.
• Standardized evaluation of sparsity efficiency: the field currently lacks a widely agreed-upon efficiency metric that jointly accounts for total parameters, active parameters, communication volume, and downstream quality, making cross-paper comparisons of "efficiency" difficult to standardize.

Addressing these challenges is likely to determine whether MoE architectures continue to scale favorably as accelerator interconnect bandwidth, memory capacity, and model sizes continue to grow at different relative rates. It is notable that several of the largest reported gains in MoE efficiency over the period surveyed in this paper — the move from top-1 to expert-choice and auxiliary-loss-free balancing, and the move from coarse-grained to fine-grained expert segmentation — came not from increasing raw expert count but from rethinking the routing and balancing algorithm itself. This suggests that continued progress on the open challenges listed above, particularly communication-aware and dynamic routing, may yield efficiency gains comparable to or exceeding those obtained from simply scaling expert count further, and that algorithmic and systems co-design will likely remain as important as architectural scale for future MoE research.

Conclusion

Mixture-of-Experts architectures have become a foundational technique for scaling large language models efficiently by decoupling total parameter count from per-token computational cost through sparse activation. This paper has surveyed the architectural foundations of MoE layers, developed a taxonomy of routing strategies ranging from simple top-1 token-choice routing to expert-choice routing and auxiliary-loss-free load balancing, and analyzed the systems-level factors of memory footprint and communication overhead that ultimately determine real-world scaling efficiency alongside raw FLOPs. Case studies of landmark systems illustrate a clear trajectory from early large-scale demonstrations toward increasingly fine-grained, sparsely activated designs paired with load-balancing mechanisms that avoid directly competing with the task loss. As accelerator hardware, interconnect bandwidth, and training infrastructure continue to evolve, routing strategy and load-balancing design — more so than expert count alone — are likely to remain the central levers by which future MoE systems achieve further gains in scaling efficiency.

Acknowledgment

The authors thank colleagues in the departmental machine learning reading group for helpful discussion during the preparation of this survey, and the anonymous reviewers whose comments improved the clarity of the presentation.

Looking forward, we expect the distinction between "architecture" and "system" to continue blurring in MoE research: routing decisions that ignore the physical cost of communication, or hardware topologies designed without regard to routing patterns, are each likely to leave efficiency on the table relative to designs that treat

routing and placement as a single, jointly optimized problem. Researchers and practitioners evaluating or adopting MoE architectures are therefore encouraged to consider expansion ratio, communication volume, and memory footprint alongside quality benchmarks when comparing systems, since any one of these factors considered in isolation can give a misleading picture of a system's true efficiency.

References

[1] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," Neural Computation, vol. 3, no. 1, pp. 79–87, 1991.
[2] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in Proc. Int. Conf. Learning Representations (ICLR), 2017.
[3] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "GShard: Scaling giant models with conditional computation and automatic sharding," in Proc. Int. Conf. Learning Representations (ICLR), 2021.
[4] W. Fedus, B. Zoph, and N. Shazeer, "Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity," arXiv:2101.03961, 2021.
[5] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, et al., "GLaM: Efficient scaling of language models with mixture-of-experts," in Proc. Int. Conf. Machine Learning (ICML), 2022.
[6] B. Zoph, I. Bello, S. Kumar, N. Du, Y. Huang, J. Dean, N. Shazeer, and W. Fedus, "ST-MoE: Designing stable and transferable sparse expert models," arXiv:2202.08906, 2022.
[7] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, et al., "Mixtral of Experts," arXiv:2401.04088, 2024.
[8] D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, et al., "DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models," arXiv:2401.06066, 2024.
[9] DeepSeek-AI, "DeepSeek-V3 technical report," arXiv:2412.19437, 2024.
[10] xAI, "Grok-1 model card," xAI Technical Documentation, 2024.
[11] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale," in Proc. Int. Conf. Machine Learning (ICML), 2022.
[12] Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. Dai, Z. Chen, Q. Le, and J. Laudon, "Mixture-of-experts with expert choice routing," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2022.
[13] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, "A survey on mixture of experts in large language models," arXiv:2407.06204, 2024.