How Mixture of Experts Models Cut the Cost of Large-Scale Inference

What Mixture of Experts Means for AI Systems Design

Large language models have grown enormously in size and capability, but running them at scale comes with steep computational costs. Each forward pass of a dense transformer activates every parameter in the network, so a 175-billion-parameter model burns through roughly the same amount of compute for a simple query as for a complex reasoning task. That is wasteful. The mixture of experts (MoE) architecture offers a different approach: instead of activating the whole network, it activates only a subset of specialized sub-networks, or experts, for each input. This design cuts the cost of large-scale inference while keeping the model's effective capacity high.

MoE is not a new idea. It was explored in the 1990s for neural networks and later adapted for sequence models. But the current wave of interest comes from its ability to scale up a model's parameter count without a proportional increase in floating-point operations per token. In AI systems design, this trade-off between model quality and inference cost is central. MoE makes it possible to serve a model with hundreds of billions of parameters on hardware that would otherwise be impractical.

How MoE Works in Practice

In a standard transformer layer, every token passes through the same feed-forward network. In an MoE layer, the feed-forward network is replaced by a set of smaller feed-forward networks (the experts) and a learned router that decides which expert(s) to send each token to. Typically, a token is routed to the top one or two experts, and the outputs of those experts are weighted and summed. The router is a small neural network that learns to map token representations to a probability distribution over experts.

This gating mechanism introduces a key challenge: load balancing. If the router sends most tokens to the same few experts, those experts become overloaded and the others remain idle, wasting capacity. To prevent this, training includes an auxiliary loss that encourages uniform routing across experts. The balance between expert specialization and uniform load is a design knob that system architects tune for their specific deployment.

Sparse Activation and Its Effect on Compute

The core efficiency gain comes from sparse activation. A dense 1-trillion-parameter model would need 1 trillion multiply-accumulate operations per token. An MoE model with the same

total parameter count but, say, 256 experts, each of 4 billion parameters, and a top-2 routing policy activates only 8 billion parameters per token. That is roughly 125 times fewer active parameters. The actual speedup depends on hardware, batch size, and communication overhead, but the principle holds: you get the representational capacity of a large model at a fraction of the per-token compute.



Memory, however, is not reduced. All experts must be loaded into memory so that any of them can be called. This means MoE models require large memory banks, often distributed across many accelerators. The trade-off is between memory footprint (high) and per-token computation (low). In AI systems design, this shifts the bottleneck from compute to memory bandwidth and inter-device communication.

Routing Strategies and Their Trade-Offs

The simplest routing strategy is top-k routing, where each token is sent to the k experts with the highest router scores. k is usually 1 or 2. Top-1 routing is the most compute-efficient but can lead to expert under-specialization because each token sees only one expert. Top-2 routing gives a richer representation and is a common sweet spot.

More advanced strategies include hash-based routing, which assigns tokens to experts deterministically based on a hash of the token ID, and learned routing with auxiliary losses that

encourage diversity. Hash routing eliminates the router parameters and the load-balancing loss, but it cannot adapt to the input distribution. Learned routing adapts but adds complexity to training. Most production MoE models use learned routing with careful tuning of the auxiliary loss coefficient.

Expert Capacity and Dropped Tokens

Each expert has a fixed capacity: the maximum number of tokens it can process in a training step or inference batch. If a batch of tokens arrives and one expert receives more tokens than its capacity, the extra tokens are dropped (or passed through a residual connection). Dropped tokens degrade model quality, so capacity must be set high enough to handle peaks. The capacity factor (actual capacity divided by expected tokens per expert) is a hyperparameter that trades off compute efficiency against quality. A capacity factor of 1.0 is efficient but risky; values of 1.25 to 2.0 are common in practice.

For inference, the batch size is often small (sometimes one token at a time), so the capacity issue is less severe. But for batched inference, especially with variable-length sequences, capacity planning becomes important. System designers must decide whether to pad sequences, group by length, or use dynamic batching to keep expert load even.

What MoE Means for the Reader

If you are designing a system that serves large models to many users, MoE can dramatically reduce your per-query compute cost. That translates to lower latency, higher throughput, or both. But the architecture introduces new considerations. Memory requirements are higher per model instance. The router adds a small overhead. And the load-balancing behavior must be monitored in production because distribution shifts can cause expert collapse over time.



One practical takeaway: MoE models are especially attractive for applications where the input distribution is broad and diverse. Because different experts specialize in different patterns, the model can handle a wide range of topics without needing to activate the entire network for each query. For a system that serves a multilingual user base, for example, some experts may specialize in languages while others handle code or technical content. The router learns to dispatch each query to the relevant experts.

Another consideration is hardware choice. MoE models benefit from high-memory devices and fast interconnects. If your inference cluster uses GPUs with limited memory, you may need to shard experts across more devices, increasing communication overhead. Some system designers choose to co-locate frequently co-activated experts on the same device to reduce network traffic. These are the kinds of architectural decisions that fall under AI systems design, and they are where the real engineering craft comes in.

Current Practice and Real-World Examples

Several large-scale deployed models use MoE. The Mixtral 8x7B model, for instance, has 47 billion total parameters but activates only 13 billion per token. It demonstrates that MoE can match or exceed the quality of a dense model with far fewer active parameters. Another example is the Gemini model family from Google, which uses MoE layers to achieve strong results across a wide set of benchmarks while keeping inference costs manageable.

For engineers building systems, the key is to evaluate MoE not just on benchmark accuracy but on the cost-quality curve for your specific workload. A dense model might be simpler to deploy and debug, but for high-volume inference, MoE can offer a clear advantage. The Futurestack.online LLM architecture discussion covers this trade-off in detail, including practical tips for configuring expert count and capacity factor based on your hardware and traffic patterns.

If you are new to MoE, start with a small model and test the router behavior on your data. Monitor expert utilization over time. If some experts become overloaded, consider adjusting the auxiliary loss coefficient or increasing the capacity factor. The Futurestack.online LLM architecture series also includes case studies on how different routing strategies affect inference latency under real-world loads, which is a useful reference when you are tuning your own system.

The Broader Implications for AI Systems Design

MoE is part of a larger trend toward conditional computation, where the model decides how much compute to allocate to each input. Other approaches include adaptive depth (early exiting) and adaptive width (skipping layers). MoE is the most mature of these techniques and is already used in production at scale. Its success has spurred research into more dynamic routing mechanisms, including learned routing that can balance specialization and load without an explicit auxiliary loss.



For system architects, the rise of MoE means that the old assumption of a dense, uniform model is no longer the only option. You can now trade memory for compute, and you can design a model that scales without a linear increase in per-token cost. This changes how you plan capacity, select hardware, and budget for inference. It also means that the line between

training and inference is blurring: the routing decisions made during training affect inference behavior, and fine-tuning an MoE model requires different techniques than fine-tuning a dense one.

The field is moving quickly. Sparse models, once a niche research topic, are now a standard tool in the AI systems design toolbox. Understanding MoE is not just about saving money on inference; it is about understanding a fundamental shift in how we build and deploy large-scale AI systems.

Futurestack.online on social media Instagram