Mixture of Experts: Making Large Language Model Inference Affordable

Large language models have grown enormously in size, and with that growth comes a steep increase in the cost of running them. Every time you ask a model to generate text, translate something, or answer a question, it must perform model inference: the forward pass through the network that turns input into output. For models with hundreds of billions of parameters, that process can be both slow and expensive. The challenge is not just about raw compute: it's about using that compute efficiently.

That is where mixture of experts (MoE) comes in. MoE is an architecture that splits a large model into many smaller, specialised sub-networks called experts, and then activates only a subset of them for each input. The result is a model that behaves like a large, knowledgeable network at inference time, but uses only a fraction of the total parameters. This makes large scale inference dramatically cheaper without sacrificing much quality. To understand how, it helps to look at what happens inside a conventional transformer during inference.

What happens during standard model inference

In a standard dense transformer, every token that passes through the model touches every parameter. The attention mechanism and the feed-forward layers all fire for every input. That means the compute cost scales with the total number of parameters, regardless of whether the task actually needs all that knowledge. A model like GPT-4 or Llama 2, with hundreds of billions of parameters, requires loading all those weights into memory and doing matrix multiplications across them for every token generated. That is why running such models on consumer hardware is nearly impossible, and why even on powerful servers the cost per query can be high.

Dense models have no way to skip irrelevant parts of their knowledge. A question about arithmetic still triggers the same layers that store facts about geography or history. The model is forced to compute everything, every time. This is fine for accuracy, but it is wasteful when the model could rely on only a few specialised regions for a given input.



How mixture of experts changes the picture

A mixture of experts model replaces the large feed-forward layers of a transformer with multiple smaller feed-forward networks, each of which is an expert. A learned gating network decides which experts to activate for each input token. Typically, only one or two experts are turned on per layer. The outputs from the active experts are combined by the gating network and passed to the next layer.

The key insight is that the experts are sparse: the model has many experts in total, but only a few are used at any one time. That means the total number of parameters loaded during inference is much smaller than the full model size. For example, a model with 64 experts and a total of 1.5 trillion parameters might only activate two experts per token, effectively using only a small fraction of its parameters for each forward pass. The rest are left idle, so they consume no compute cycles.

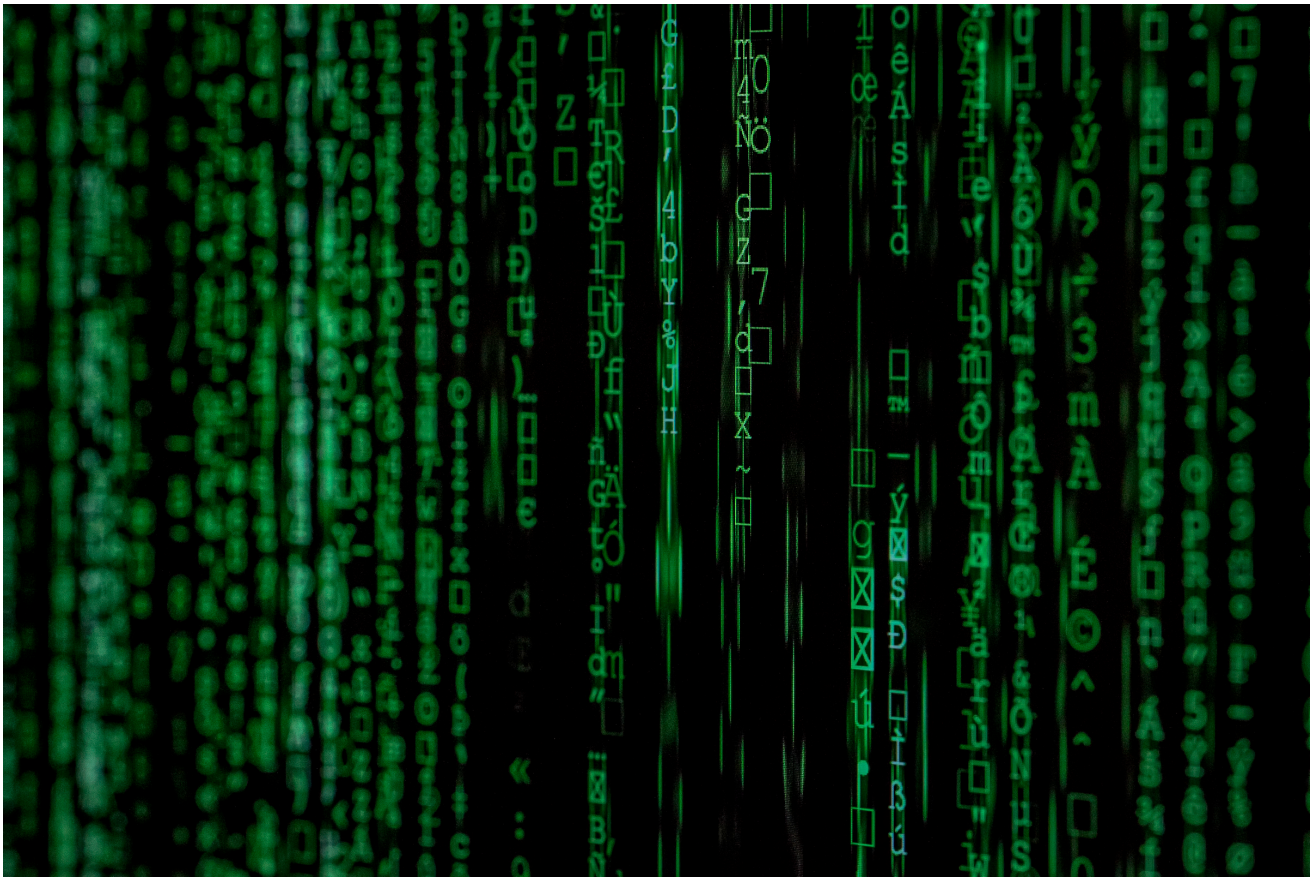
This sparse activation has a direct effect on cost. Since the bulk of the computation happens inside the active experts, and since those experts are small relative to the whole model, the inference cost per token is much lower than it would be for an equally capable dense model. The memory footprint also shrinks because only the experts that are currently needed have to be kept in fast memory, while the rest can stay on slower storage or be swapped in as needed.

Routing and load balancing: the glue that makes it work

The gating network, often called a router, is a small learned module that sits between layers. It takes the token embedding and computes a score for each expert. The top-k experts with the highest scores are selected, and their outputs are weighted by the gating values. The router must learn to assign tokens to experts in a way that makes sense, sending mathematical queries to a compute expert and factual queries to a knowledge expert.

But there is a catch. If the router is not careful, it can overload a few experts while leaving others idle. That would defeat the purpose of sparsity, because the busy experts would become bottlenecks. To prevent this, MoE models use load balancing loss during training. This loss penalises the model when the distribution of tokens across experts becomes too uneven. The result is a router that spreads tokens roughly evenly across experts, even though each token only activates a few.

Good load balancing also reduces the variance in inference latency. When experts are evenly utilised, the system can predict how long a forward pass will take, which is important for production services that need stable response times.



Trade-offs: quality, training cost, and engineering complexity

Mixture of experts is not a free lunch. The quality of the model depends on how well the experts specialise. If the experts learn overlapping representations, the model may not benefit from sparsity. In practice, MoE models often match or even exceed the quality of dense models of similar total parameter count, but they do require careful tuning of the router and the load balancing loss.

Training an MoE model is more complex than training a dense model. The experts can be placed on different devices, and communication between them becomes a bottleneck. Techniques like expert parallelism and all-to-all communication are needed to keep training efficient. For companies that already have a dense training pipeline, switching to MoE means redesigning the infrastructure. However, once the model is trained, the inference cost savings are large enough to justify the upfront investment.

Another trade-off is that MoE models have a larger total parameter count than their dense counterparts for the same effective compute. That means they require more storage space. But since most parameters remain idle during inference, the storage cost is often a one-time expense that is dwarfed by the ongoing inference savings. For large scale deployments, where inference runs continuously, the reduction in compute costs easily outweighs the extra storage.

What this means for practitioners

For anyone building applications that rely on large language models, MoE offers a way to run bigger, more capable models without a proportional increase in inference budget. A single MoE model can replace a dense model that is several times larger in active parameters, giving better performance for the same cost. This is especially useful for products that need to serve many users simultaneously, where the cost per query directly affects margins.

It also opens the door to deploying models on less expensive hardware. Because the active parameter count is low, MoE models can run on GPUs with smaller memory, or even on CPUs with efficient sparse computation libraries. The trade-off is that the model must be served with an inference engine that supports sparse routing, and not every framework does. But as MoE becomes more popular, support is growing across major platforms.

Readers interested in the latest developments in MoE and other inference optimisations can follow coverage on Futurestack.online, which regularly explores how distributed systems and novel architectures are reshaping the cost profile of AI. The journal covers the engineering side of these advances, from training pipelines to deployment strategies, and provides practical insight for those building production systems. Futurestack.online also dives into the broader implications of sparsity, including how it relates to agentic software and cloud architecture.



Looking ahead

Mixture of experts is not a new idea, but it has found a natural home in large language models. As models continue to grow, the pressure to keep inference affordable will only increase. Techniques like MoE, along with other sparsity methods, will become standard tools in the engineer's kit. They allow the field to push the frontier of model capability while keeping the cost of deployment within reach.

The next few years will likely see even more sophisticated routing strategies, perhaps learned jointly with the model, and better hardware support for sparse operations. For now, MoE stands as one of the most effective ways to reduce the cost of large scale model inference. It is a reminder that sometimes the smartest way to use a large system is not to use all of it at once.

Futurestack.online on social media Instagram