MoE Models Explained: Why Mixtral Uses 46B Parameters But Runs Like 13B

February 23, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Mixture of Experts (MoE) models contain multiple expert sub-networks but only activate a fraction per token. Mixtral 8x7B has 46.7B total parameters but activates ~13B per token, so inference runs at 13B speed. The catch: you still load ALL 46.7B params into VRAM. 'Runs like 13B' means speed, not memory. Most people on 24GB GPUs are better off with a dense model like Qwen 3 32B at Q4.

📚 **More on this topic:** Mixtral VRAM Requirements · Quantization Explained · DeepSeek Models Guide · Planning Tool

Mixtral 8x7B has 46.7 billion parameters but runs at the speed of a 13B model. That single sentence has confused more people than any other fact in local AI.

The confusion is understandable. “Runs like 13B” sounds like it needs 13B-level resources. It doesn’t. You need enough VRAM to hold all 46.7 billion parameters — the same as any dense 46B model. The speed benefit is real. The memory savings are not.

This is the Mixture of Experts architecture, and understanding it properly will save you from the most common hardware mistake in local AI: downloading an MoE model that doesn’t fit on your GPU.

What mixture of experts actually is

A standard “dense” model, like Llama 3.3 70B or Qwen 3 32B, activates every single parameter on every single token. If it has 70 billion parameters, all 70 billion do work each time the model processes a word.

MoE takes a different approach. Instead of one massive feed-forward network (FFN) layer, an MoE model has multiple smaller FFN layers called **experts**. A lightweight **gating network** (also called a router) looks at each incoming token and decides which experts should handle it. Only the selected experts activate. The rest sit idle.

Think of it like a hospital. A dense model is a single doctor who handles everything (cardiology, neurology, orthopedics) for every patient. An MoE model is a hospital with eight specialist doctors. Each patient only sees two of them, but the hospital still needs to pay all eight salaries and maintain all eight offices.

The core components

Every MoE model has three parts:

1. **Shared layers**: attention layers, embeddings, and layer norms that process every token (same as a dense model)
2. **Expert FFN blocks**: multiple parallel feed-forward networks, only some activated per token
3. **Gating network**: a small learned router that scores each expert for each token and picks the top-k

The gating network is tiny, typically a single linear layer per MoE block. It takes the hidden state of the current token, produces a score for each expert, applies a softmax, and routes the token to the top-k experts. The outputs from the selected experts are combined using the gating scores as weights.

```
Token → Attention (shared) → Gating Network → [Expert 2, Expert 5] → Weighted Sum → Output
                                               (6 experts idle)
```

This routing happens independently for every token. One word might use experts 1 and 4. The next might use experts 3 and 7. The gating network learns during training which experts specialize in what, though the specialization is emergent, not manually designed.

Mixtral 8x7B: the concrete example

Mixtral 8x7B from Mistral AI is the model that brought MoE to consumer-accessible local AI. It's the clearest example to study.

The numbers

Component	Value
Total parameters	46.7B
Number of experts	8 (per MoE layer)
Experts active per token	2
Active parameters per token	~12.9B

Component	Value
Shared attention parameters	~6.4B
Expert FFN parameters	~40.3B (8 × ~5B each)
Context window	32K tokens

Each of the 32 transformer layers has 8 expert FFN blocks. On every token, the gating network picks the 2 best-matching experts. So you compute the full attention stack (~6.4B params) plus 2 out of 8 expert blocks (~10.5B params), totaling roughly 12.9B active parameters per forward pass.

Why it’s fast

Inference speed is dominated by how many parameters get computed per token. Mixtral computes ~13B parameters per token. A dense 13B model computes ~13B parameters per token. So they run at roughly the same speed, about 30-40 tokens/second on an RTX 3090 at Q4 quantization.

This is the MoE win: you get the quality of a model trained with 46.7B parameters, at the speed of a 13B model. During training, all those expert parameters learned from the data. The model has the knowledge capacity of 46.7B params. It just accesses that knowledge selectively.

The VRAM trap

MoE models need VRAM proportional to TOTAL parameters, not active parameters.

When you load Mixtral 8x7B, all 46.7 billion parameters go into memory. All 8 experts in every layer must be resident in VRAM, even though only 2 are used per token. Why? Because the gating network decides at runtime which 2 experts each token needs. The model can’t know in advance which experts will be called, so all of them must be ready.

The math that catches people

Here’s the scenario that plays out daily in r/LocalLLaMA:

1. Someone reads “Mixtral 8x7B runs at 13B speed”
2. They check their 16GB GPU and think “I can run 13B models, so Mixtral should fit”
3. They download the Q4 GGUF (about 26GB)

4. They get an out-of-memory error
5. They're confused because "it's supposed to be like a 13B model"

It is not like a 13B model for memory. It is like a 46.7B model for memory. The Q4 quantized version needs approximately 26GB of VRAM, the same as any dense 46B model at Q4. At FP16, it needs about 93GB.

To put it bluntly:

What "Runs Like 13B" Means	What It Does NOT Mean
Inference speed of ~13B	VRAM usage of ~13B
Tokens per second of ~13B	Download size of ~13B
Compute per token of ~13B	Disk space of ~13B

A concrete comparison

Let's say you have an RTX 4090 with 24GB VRAM:

- **Qwen 3 32B at Q4_K_M**: ~20GB VRAM. Fits. All 32B parameters active per token. Excellent quality.
- **Mixtral 8x7B at Q4_K_M**: ~26GB VRAM. Does NOT fit in 24GB. Only ~13B active per token.

The dense 32B model that uses every parameter fits on your GPU. The MoE model that only uses 13B parameters per token does not fit. This is the VRAM trap. The model that computes fewer parameters per token actually needs more memory.

Our Planning Tool correctly uses total parameters for MoE VRAM estimates, not active parameters. If your planner or calculator uses active params, throw it out.

MoE models available today

Here's every MoE model you might consider running locally:

Model	Total Params	Active Params	Experts (Total/ Active)	Context	License	Reality Check
Mixtral 8x7B	46.7B	~13B	8/2	32K	Apache 2.0	

Model	Total Params	Active Params	Experts (Total/ Active)	Context	License	Reality Check
						The OG consumer MoE. Needs ~26GB at Q4.
Mixtral 8x22B	141B	~39B	8/2	64K	Apache 2.0	Serious hardware. ~66GB at Q4.
DeepSeek V3	671B	~37B	256/8	128K	MIT	Multi-node territory. ~350GB at Q4.
DeepSeek R1	671B	~37B	256/8	128K	MIT	Reasoning variant of V3. Same hardware needs.
DBRX	132B	~36B	16/4	32K	Databricks Open	16 experts, 4 active. ~66GB at Q4.
Llama 4 Scout	109B	~17B	16/1	10M	Llama 4	~55GB at Q4. Full guide here.
Llama 4 Maverick	400B	~17B	128/1	1M	Llama 4	~200GB at Q4. Datacenter-class.
Qwen 1.5 MoE A2.7B	14.3B	~2.7B	60/4	32K	Qwen	Small MoE experiment. ~8GB at Q4. Niche.

Notice the pattern: total parameters drive VRAM, not active parameters. DeepSeek V3 activates only 37B params per token (fewer than a dense 70B model) but needs 350GB at Q4. That's more VRAM than most servers have.

For most of these models, the DeepSeek Models Guide and Mistral & Mixtral Guide have detailed setup instructions.

VRAM requirements: Mixtral at every quantization

Since Mixtral models are the most commonly run MoE models locally, here are precise VRAM estimates. These include model weights plus a reasonable KV cache overhead for 4K context.

Mixtral 8x7B (46.7B total)

Quantization	Model Size	VRAM Needed	Fits On
Q2_K	~17 GB	~19 GB	RTX 3090 24GB (tight)
Q3_K_M	~21 GB	~23 GB	RTX 3090 24GB (tight)
Q4_K_M	~26 GB	~28 GB	2x RTX 3060 12GB, Mac 32GB+
Q5_K_M	~32 GB	~34 GB	Mac 36GB+, 2x 24GB GPUs
Q6_K	~38 GB	~40 GB	Mac 48GB+, 2x 24GB GPUs
Q8_0	~49 GB	~52 GB	Mac 64GB+, A6000 48GB (offload)
FP16	~93 GB	~96 GB	2x A100, Mac 128GB

Mixtral 8x22B (141B total)

Quantization	Model Size	VRAM Needed	Fits On
Q2_K	~52 GB	~56 GB	2x RTX 3090, Mac 64GB+
Q3_K_M	~63 GB	~67 GB	Mac 96GB+, A100 80GB
Q4_K_M	~80 GB	~85 GB	Mac 96GB+, A100 80GB (tight)
Q5_K_M	~96 GB	~101 GB	Mac 128GB, 2x A100
Q6_K	~112 GB	~118 GB	Mac 128GB+, 2x A100
Q8_0	~150 GB	~156 GB	Mac 192GB, multi-A100
FP16	~282 GB	~290 GB	Multi-H100 cluster

For full breakdowns including offloading strategies, see the dedicated Mixtral VRAM Requirements guide.

MoE vs. dense: what to run at each VRAM budget

This is the practical table. For each VRAM tier, here’s what actually makes sense.

VRAM	Best Dense Option	Best MoE Option	Winner	Why
8 GB	Llama 3.1 8B Q4 or Qwen 3 8B Q4	Qwen 1.5 MoE A2.7B Q4 (~8GB)	**Dense**	The MoE is tiny and underperforms 8B dense models
12 GB	Qwen 3 14B Q4 (~9GB)	None that fit well	**Dense**	No competitive MoE fits in 12GB
16 GB	Qwen 3 32B Q3 (~16GB)	None that fit well	**Dense**	Still no good MoE options at this tier
24 GB	Qwen 3 32B Q4 (~20GB)	Mixtral 8x7B Q2 (~19GB)	**Dense**	Qwen 3 32B outperforms Mixtral on every benchmark. Not close.
48 GB	Llama 3.3 70B Q4 (~42GB)	Mixtral 8x22B Q3 (~67GB, too big); Mixtral 8x7B Q8 (~52GB, barely fits)	**Dense**	70B dense still beats Mixtral 8x22B which doesn't even fit. Mixtral 8x7B at Q8 fits but loses to 70B on quality.
80 GB	Llama 3.3 70B Q8 (~74GB)	Mixtral 8x22B Q4 (~85GB, tight)	**Dense / Tie**	At 80GB, you can run either 70B dense at near-lossless quant or 8x22B at Q4. Similar quality, 70B is the safer pick.
2x 48 GB	Llama 3.3 70B FP16 (~140GB)	Mixtral 8x22B Q5 (~101GB)	**Depends**	8x22B at high quant is compelling here. But consider DeepSeek distilled models too.

The pattern is clear: **below 48GB, dense models win.** MoE only becomes attractive when you have enough VRAM to load the full model at decent quantization AND the MoE model's total knowledge capacity outweighs what a dense model can offer at the same memory footprint.

For consumer hardware (the single 24GB GPU that most of our readers have), MoE models are almost never the right choice.

How routing actually works

The gating network is simple but important. Here's what happens at each MoE layer:

```
# Simplified pseudocode for MoE routing
def moe_forward(hidden_state, experts, gate):
    # Gate produces a score for each expert
    scores = gate(hidden_state)  # shape: [num_experts]

    # Softmax to normalize
    probs = softmax(scores)

    # Select top-k experts (k=2 for Mixtral)
    top_k_indices = topk(probs, k=2)
    top_k_weights = probs[top_k_indices]

    # Renormalize weights so they sum to 1
    top_k_weights = top_k_weights / sum(top_k_weights)

    # Run token through selected experts and combine
    output = sum(weight * experts[idx](hidden_state)
                 for idx, weight in zip(top_k_indices, top_k_weights))

    return output
```

The gate itself is typically just a linear projection: `gate_scores = W_gate @ hidden_state`, where `W_gate` has shape `[num_experts, hidden_dim]`. For Mixtral with 8 experts and a hidden dimension of 4096, that's a 8 x 4096 = 32,768 parameter matrix, negligible compared to the expert weights.

Load balancing

A naive gating network might route most tokens to the same 2 "favorite" experts, leaving 6 experts undertrained. To prevent this, MoE training includes an **auxiliary load balancing loss** that penalizes uneven expert utilization. The original Mixtral paper and the Switch Transformer paper cover this in detail.

In practice, well-trained MoE models distribute tokens roughly evenly across experts, though some experts do develop soft specializations. Research on Mixtral shows that certain experts handle particular languages or domains more often, but no expert is exclusively specialized. Every expert can handle general text.

Why same-family experts specialize

The experts aren't pre-assigned roles. They start as identical random initializations and differentiate during training through the combination of the gating network's routing decisions and the load balancing loss. It's similar to how neurons in a neural network develop feature selectivity: emergent specialization, not designed specialization.

This means you can't just grab one expert and use it as a standalone model. Each expert is trained in the context of the routing mechanism and the shared attention layers. They're collaborative specialists, not independent models.

When MoE actually wins

MoE is a good architecture stuck on the wrong hardware. Consumer GPUs don't have the VRAM to make it worthwhile yet. In a datacenter context, it's genuinely superior:

MoE advantages

MoE models learn more per training FLOP because the full parameter count sees data during training, but each training step only computes a fraction. DeepSeek V3 was reportedly trained for $5.5M, which is remarkably cheap for a 671B parameter model.

The per-token compute savings also translate directly to lower cost per query in datacenter deployments serving thousands of users. This is why DeepSeek, Google (Gemini uses MoE), and others chose this architecture for production APIs.

On top of the cost savings, MoE gives you a higher quality ceiling. Given the same inference compute budget, an MoE model can be smarter than a dense model because it has more total parameters to store knowledge in. A 46.7B-parameter MoE that runs at 13B speed will generally outperform a 13B dense model trained on the same data.

When dense wins for local AI

If you have 24GB VRAM, you can fit a dense 32B model at Q4 or an MoE 46.7B model at Q2. The dense model at higher quantization will almost certainly perform better because aggressive quantization destroys MoE routing precision.

Dense models are also easier to fine-tune, quantize predictably, and don't have routing overhead. GGUF quantization for MoE models requires expert-aware handling that's well-supported in llama.cpp now but was buggy for months after Mixtral's release.

And dense models degrade more gracefully under quantization. MoE models have the gating network and expert weights both getting quantized, and errors in gating decisions cascade. Sending a token to the wrong expert entirely is worse than slightly imprecise math in the right expert.

The practical calculus

Here's the decision framework.

24GB VRAM or less

Run dense models. Period. Qwen 3 32B at Q4 is better than any MoE model you can squeeze into 24GB. If you're on 16GB or 12GB, see our guides for 16GB and 12GB recommendations.

The only exception: if you specifically need the Mixtral fine-tune ecosystem (there are specialized Mixtral fine-tunes for certain tasks), and you're willing to run Q2_K quantization with degraded quality.

48GB VRAM

Mixtral 8x22B fits at Q3 quantization (~67GB with offloading). But Llama 3.3 70B at Q4 (~42GB) on a single 48GB GPU will outperform it in most use cases with simpler setup. MoE becomes interesting here only for specific tasks where 8x22B's training diversity helps.

80GB+ VRAM

Now MoE models make real sense. Mixtral 8x22B at Q4-Q5 quantization on an A100 80GB gives you excellent quality. DeepSeek R1 distilled variants (dense) are also compelling at this tier, though.

Multi-GPU / 128GB+ Mac

This is MoE's sweet spot for local AI. Mixtral 8x22B at Q6+ or even Q8 quantization. If you have the memory, MoE delivers: fast inference with high knowledge capacity. The full DeepSeek V3 at Q4 needs ~350GB, but if you have the hardware, nothing else touches its quality.

What to remember

MoE gives you the speed of a small model but the memory footprint of a big one. Never confuse inference speed with VRAM requirements. Mixtral 8x7B (46.7B total) needs the same memory as a dense 46B model, no exceptions.

Below 24GB VRAM, skip MoE entirely. Dense models at the same memory budget will outperform them. Above 80GB, MoE starts to pay off with faster inference than a dense model of equivalent quality.

DeepSeek, Google Gemini, and Llama 4 all chose MoE. It's the dominant architecture for frontier models now, and as consumer VRAM grows, MoE will eventually become the default local choice too.

If you have a single 24GB GPU, run Qwen 3 32B and forget about MoE. When 48GB GPUs hit consumer prices, or when you build a multi-GPU setup, come back and revisit.

Related guides

- Mixtral 8x7B & 8x22B VRAM Requirements: precise VRAM tables for every quantization level
- LLM Quantization Explained: how Q4, Q5, Q8 affect model quality and size
- DeepSeek Models Guide: V3, R1, and the distilled dense variants
- VRAM Requirements for Local LLMs: complete reference for every popular model

---

Source: https://insiderllm.com/guides/moe-models-explained/

Free guides for running AI locally