Deepseek Model Guide

baseten

Table of Contents

PART 01 **Introduction**

PART 02 **What are the cDeepSeek models?**

- What makes DeepSeek unique?
- Popular DeepSeek models
- DeepSeek in the LLM landscape

PART 03 **Why deploying DeepSeek is challenging**

- Infrastructure challenges
  - DeepSeek model parallelism
  - Serving DeepSeek on B200s and H200s
  - Serving DeepSeek on H100s
  - B200 vs. H200 vs. H100 for DeepSeek model serving
- Model performance challenges
  - DeepSeek inference metrics to track
  - Quantization
  - Inference frameworks
  - Caching
  - Decoding strategies

PART 04 **Deploying, optimizing, and scaling DeepSeek in production**

- Deploying DeepSeek in production
  - DeepSeek security and compliance
- Scaling DeepSeek in production
- Optimizing DeepSeek in production

PART 05 **Conclusion**

PART 01

The complete DeepSeek model guide:

Deploying, scaling, and optimizing DeepSeek in production

When DeepSeek V3 and R1 dropped, they showed AI builders that they can get the quality of closed-source models—comparable to GPT-5, GPT-4o, and Claude 3—at a fraction of the cost, and with more control over what happens with their data. But while DeepSeek closed the gap on open-source vs. closed-source model quality, it left another one: how to deploy, scale, and optimize the DeepSeek models in production.

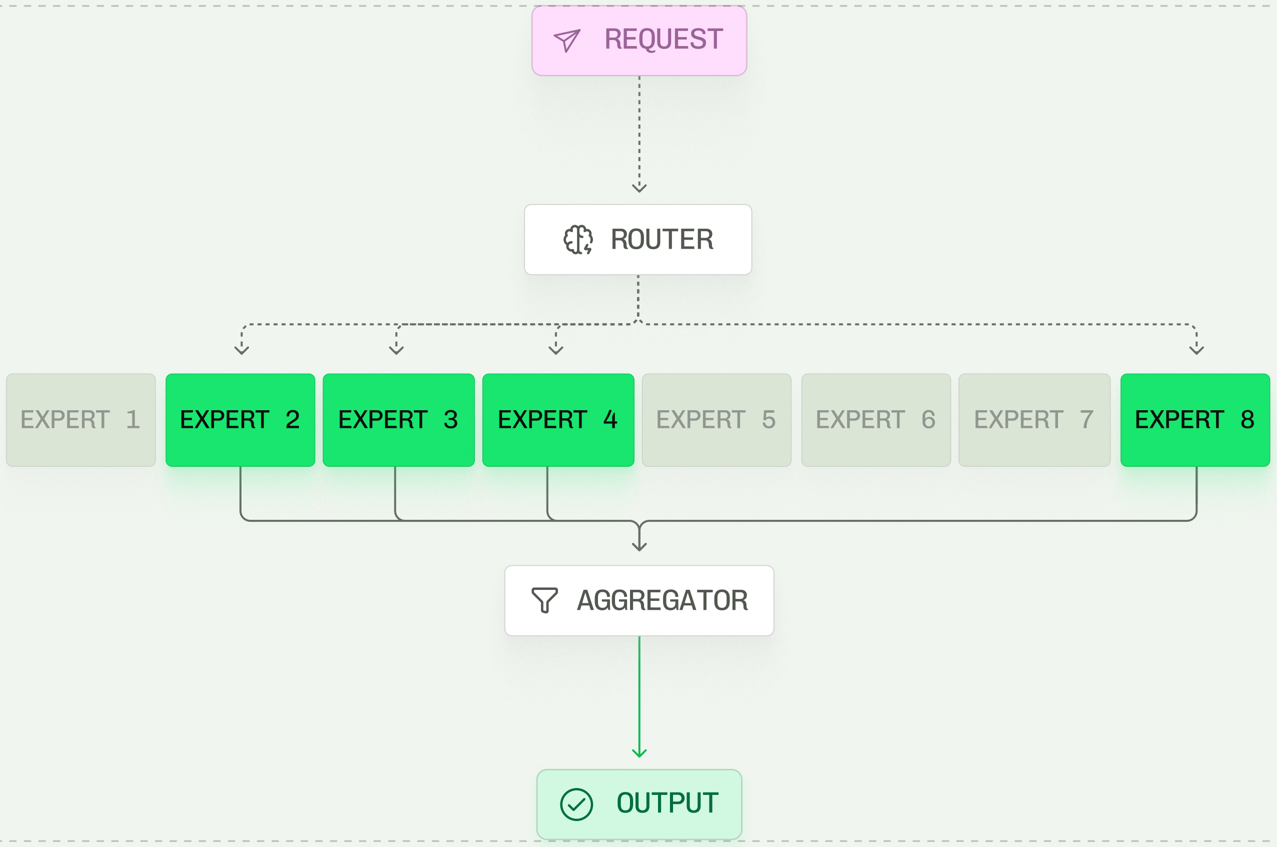
In this guide, we'll cover exactly that.

We'll start by looking at what makes the DeepSeek models special, yet difficult to serve. We'll look at the core challenges facing DeepSeek deployments at the infrastructure and runtime layers, as well as the different tools and techniques you can use to solve them. Finally, we'll cover what a production-ready setup looks like for serving DeepSeek performantly, reliably, and securely at scale.

What are the DeepSeek Models?

DeepSeek is a model family of open-source large language models (LLMs). The DeepSeek model family became extraordinarily popular with the release of DeepSeek V3, their flagship model, and DeepSeek R1, a fine-tuned version of V3 optimized for reasoning and chat.

Part of the innovation behind DeepSeek V3 and DeepSeek R1 is how they're built, using a Mixture of Experts (MoE) architecture. MoE is an architecture where the larger model is composed of many smaller "expert" models, each specialized in a different task. A lightweight router network determines which experts are most relevant to a given request and activates them accordingly.



CONTINUED >

What are the DeepSeek Models?

While DeepSeek didn't invent MoE, they certainly popularized it. So while the full model is huge (671B), only a fraction of it is typically used during inference (like 37B parameters). That said, at large batch sizes more experts get activated; given a high amount of request diversity in a batch, all of the experts could realistically get activated during inference.

DeepSeek V2 was the company's first large-scale MoE model (236B total parameters, ~21B active), showcasing strong performance for its time but not yet reaching closed-source quality. DeepSeek V3 expanded the architecture with 671B total parameters, a 128K context window (the number of words or characters it can retain when generating a response), better expert balancing, and higher overall quality.

What makes DeepSeek unique?

DeepSeek's claim to fame was its ability to offer closed-source performance (competitive with GPT-5, o3, and Claude 4 Sonnet) at a fraction of the price. The cost savings mainly come from its efficient MoE architecture and the fact that, by being totally open-source, you can host it yourself without paying a closed-source markup.

Since the release of V3 and R1, enterprises and startups have started building their AI strategies around using these models.

Model	What it is	Use cases	Comparable closed models	Notable variants
DeepSeek-V3	High-performance general LLM	Chat, math, translation, and tool use	GPT-40 Claude 3.5 Sonnet	V3-0324 March '25: Improved function calling and overall performance
DeepSeek-R1	Fine tuned from V3 for reasoning and chatbot alignment	Assistant-style tasks and reasoning-heavy applications	OpenAI o1 and o3 Gemini 2.5 Pro	R-0528 May '25: Improved reasoning, coding, and reduced hallucinations
DeekSeek-Coder	Trained specifically for code use cases	Code gen and debugging across 300+ programming languages	GPT-4 Turbo Claude 3 Sonnet CodeWhisperer	Coder v1 dense 1.3B-33B Coder v2 MoE, 128 context 236B total
R1-Distill Models	Lightweight, open-source reasoning models distilled from R1	Cost-sensitive and edge use cases	N/A	Qwen 1.5B/7B/14B/32B Llama 8B/70B

When we refer to DeepSeek, we're referring to the flagship models (V3/R1) unless otherwise specified.

DeepSeek in the LLM landscape

DeepSeek in the LLM landscape

In terms of comparable closed-source models, the latest releases of DeepSeek V3.1-0324 outperforms GPT-4o on various benchmarks, with comparable performance to Claude 4 Sonnet. DeepSeek R1 performs neck-to-neck on various benchmarks compared to OpenAI o1, with the R1-0528 update achieving comparable performance to o3 and Gemini 2.5 Pro.

In the open-source landscape, V3 and R1 outperform leading models like Llama 3 70B and Qwen3 225B while maintaining a larger context window of 128K tokens. R1-0528 is competitive with models like Kimi K2 and GLM-4.5 on code generation and agentic tasks. DeepSeek V3.1 delivers broader reasoning depth with strong coding results compared to GLM-4.5, although the latter may be more cost-effective given its smaller size (355B total, 32B active).

To provide a more efficient alternative to the full-fledged 671B models, DeepSeek also released several distilled models; smaller, open-source models trained to replicate the reasoning abilities of DeepSeek-R1. These models are fine-tuned on top of existing, popular open-source models like the Qwen model family and Llama model family, with sizes ranging from 1.5B to 70B parameters.

For longer context windows, Llama 4 Scout (10M token context) and Maverick (1M token context) are good alternatives. That said, the DeepSeek models are excellent all-rounders for deeper reasoning and tool use.

However, achieving this level of performance (while having full control over your data) first requires serving DeepSeek in production. While the MoE architecture makes the large models more efficient, they're still difficult to load into memory, and making their performance acceptable for user-facing applications takes a lot more than just raw GPUs.

Why deploying DeepSeek is challenging

At 671 billion parameters, the size of DeepSeek V3 and R1 makes them difficult to deploy for two reasons:

1. You need enough GPU memory to load the models
2. You need to use different optimizations to get performance good enough to be user-facing

The first is an infrastructure issue; the second is a runtime issue.

Infrastructure challenges

First: the memory requirements. In FP8, each parameter requires 1 GB of VRAM to load. Plus, you'll want to reserve a substantial amount of GPU memory for the KV cache (a few 100 GB extra).

Even on the latest Blackwell-series GPUs (B200s), DeepSeek V3 and R1 can't fit on a single GPU. To run DeepSeek across a single GPU node (consisting of 8 GPUs), you'll need to leverage model parallelism techniques.

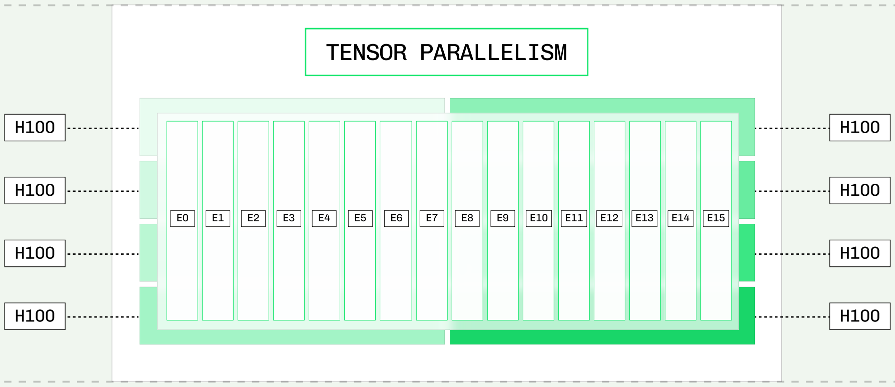
DeepSeek model parallelism

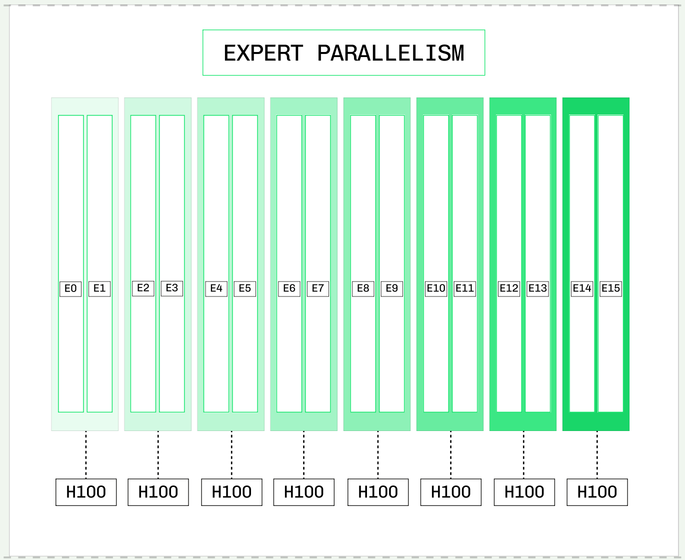
Model parallelism splits a single AI model across multiple GPUs. Two such techniques are tensor and expert parallelism:

- Tensor parallelism splits tensor operations (like matrix multiplications) across GPUs to enable faster computation for large layers.
- Expert parallelism distributes different MoE experts across GPUs, so only part of the weights need to be loaded per GPU.

CONTINUED ›

Infrastructure challenges





So while expert parallelism puts different experts on specific GPUs, tensor parallelism takes tensors (like from the model weights) and splits them across multiple GPUs. With large models like DeepSeek, matrix multiplications can become a bottleneck—tensor parallelism addresses this bottleneck by throwing more compute at the problem, helping lower latency.

While tensor parallelism improves latency, it does introduce some overhead since matrix multiplications require additional cross-GPU communication. Expert parallelism improves throughput, making it useful for workloads with high batch volume. For instance, on an 8-GPU node, you can use tensor parallelism to run eight concurrent matrix multiplications in parallel. Similarly, you can split DeepSeek V3.1 and R1's 256 total expert models so that you have 32 experts loaded per GPU.

CONTINUED >

Infrastructure challenges

We decide when and how to leverage these model parallelism techniques based on target metrics and requirements on a per-deployment basis. These model parallelism techniques introduce some orchestration overhead, especially in terms of load balancing, routing (tokens to experts), and inter-GPU communication. But once implemented, using these techniques, you can serve DeepSeek on more powerful Blackwell- and Hopper-series GPU nodes.

Serving DeepSeek on B200s and H200s

Ideally, you'd use B200 GPUs to deploy DeepSeek (that's what we typically use at Baseten). With B200s, you easily have enough memory to load DeepSeek within a single node, and they give the best performance out of the box — with over 3x higher throughput compared to H200s.

B200s are technically more expensive than alternatives (H200s and H100s), but the performance improvements they offer make them much more cost-efficient (if you can process twice as many requests with B200s, you'll only need half as many for your workload). That said, B200 GPUs are not easy to come by. While we can source B200s across clouds due to our Multi-cloud Capacity Manager (MCM), typically they're very limited in supply; without something like MCM, scaling DeepSeek on B200s can be difficult or impossible.

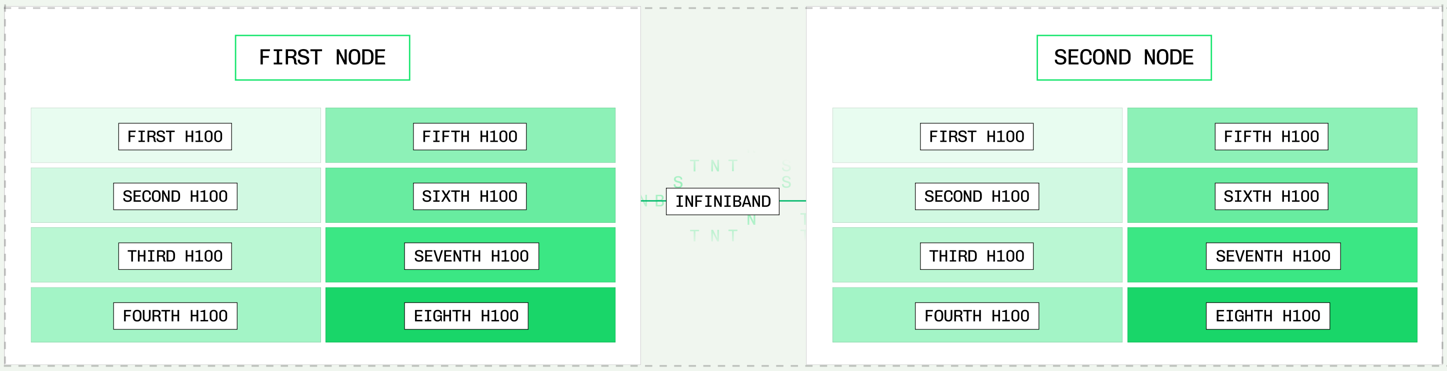
Because of this, many people use H200s or H100s to deploy DeepSeek. A single H200 has 141 GB of memory, meaning a node (8 H200s) has 1.13 TB of VRAM — plenty to fit DeepSeek V3 or R1. H200s aren't as performant as B200s, but they're still performant enough for most use cases.

Serving DeepSeek on H100s

Between B200s, H200s, and H100s, H100s are the easiest to come by. But with only 80 GB per GPU (640 GB per node), DeepSeek won't fit on a single node. To run DeepSeek on H100s, you'll need to leverage multi-node inference, which splits the model across two 8xH100 nodes.

CONTINUED ›

Infrastructure challenges



Multi-node inference introduces additional challenges around ensuring efficient communication between nodes and provisioning GPUs with high-bandwidth interconnects between nodes. That said, if you want to serve DeepSeek on H100 GPUs, multi-node inference is the only way to do it.

You can find a summary of GPU choices for DeepSeek deployments in the table below.

B200 VS. H200 VS. H100 FOR DEEPSEEK MODEL SERVING

GPU type	Memory per GPU	Memory per node	DeepSeek fits on a single node?	Advantages	Disadvantages
B200	192GB	1.5TB	Yes	Most performant	Hardest to source
H200	141GB	1.13TB	Yes	Good performance	Difficult to source
H100	80GB	640GB	No	Easiest to source Cheapest option	Least performant Multi-node inference

Once you have the infrastructure to deploy and scale DeepSeek, you'll need to turn your attention to its performance. DeepSeek performance is influenced not only by hardware choice but by factors like the model quantization, inference framework, caching, and decoding approach.

But to know how effective these techniques are, you'll first need to be tracking the right metrics.

Infrastructure challenges

DeepSeek inference metrics to track

For production applications like chat, agents, and code generation, the most important inference metrics to track will be latency and throughput. Specifically:

- Time to first token (TTFT): Measures the delay between input submission and the first token generated from the response. Typically measured in milliseconds.
- End-to-end latency: Total time from prompt input to full response output. Typically measured in milliseconds.
- Throughput: Average output speed measured in tokens per second.

TTFT is important for perceived latency because users can already see the response being generated. Throughput determines how many tokens per second your system can generate across requests, which affects how many concurrent users you can serve without degrading latency.

Quantization

Given how large DeepSeek V3 and R1 are, quantization is crucial for improving DeepSeek model performance. Quantization maps the model weights to a number format that uses fewer bytes per parameter, reducing their memory footprint.

DeepSeek V3 and R1 are originally in FP16 precision, but the current best practice is to use FP4 quantization for the model weights and FP8 for the attention (key-value, or KV) layer. During the prefill phase, the model builds up context in the KV cache, and during decoding, that cached context is used to generate output tokens. The attention (or KV) layer sits between the prefill and decoding, determining which experts get activated. Using FP8 for this layer preserves a higher precision for routing and decoding, while using FP4 for the remaining weights significantly reduces the overall memory footprint.

CONTINUED ›

Model Performance Challenges

Full FP4 quantization is not currently supported by NVIDIA's TensorRT-LLM package, but if added in future releases, it could lead to further speed gains (assuming a negligible impact on model quality). Until then, this mixed scheme provides the best balance between accuracy and performance for large MoE inference, leading to a lower TTFT and end-to-end latency and higher model throughput.

Inference frameworks: SGLang vs. TensorRT-LLM vs. vLLM for DeepSeek model serving

The inference framework you use will affect model throughput, end-to-end latency, and TTFT. SGLang and TensorRT-LLM support DeepSeek V3 and R1 out of the box, although initial support took about two weeks from their initial release. Both packages offer several integrations tailored to their MoE architecture, including support for mixed-precision FP8/FP4 quantization and expert routing.

TensorRT-LLM is specialized for NVIDIA GPUs, and our team has worked closely with NVIDIA to contribute patches improving performance for DeepSeek. As a result, we've found TensorRT-LLM to be the most robust option for serving DeepSeek models on Hopper- and Blackwell-series GPUs. TensorRT-LLM also includes custom kernels and advanced speculation support for additional performance gains.

vLLM has limited support for DeepSeek's full feature set (like max context length, expert parallelism, and tool calling) and lower performance, so we don't recommend using vLLM to serve DeepSeek in production.

Model Performance Challenges

Caching

Caching is critical to optimizing DeepSeek inference performance, particularly for reducing TTFT.

The central caching mechanism we use is KV cache reuse, where prefilled key/value attention blocks from previous requests are reused for new requests that share the same input prefix. This eliminates the need to re-run the prefill phase, which is one of the most expensive stages in inference for long prompts, thereby significantly reducing TTFT.

To maximize the effectiveness of this caching strategy across distributed hardware systems, you can use KV cache-aware routing, where incoming requests are routed to the same node that previously served a similar (or identical) prefix. This is especially beneficial in RAG pipelines, chat systems with repeated instruction headers, or for prompt templates reused across sessions.

Decoding strategies

Speculative decoding is a technique used to speed up LLM inference by using a smaller, faster draft model to predict multiple tokens in parallel, which the larger, more accurate LLM then verifies.

For DeepSeek, we use speculative decoding along with multi-token prediction (MTP), implemented via an MTP head attached to the end of the model. The MTP head predicts a configurable number of candidate tokens per pass (versus just generating one token per forward pass, as is otherwise standard). Even if some tokens must later be verified or corrected (e.g., via speculative decoding), the net cost is typically much lower than fully decoding one token at a time, since it reduces the number of total forward passes through the model.

Speculative decoding and MTP lead to lower end-to-end latency and GPU overhead, as well as higher throughput, especially for longer outputs.

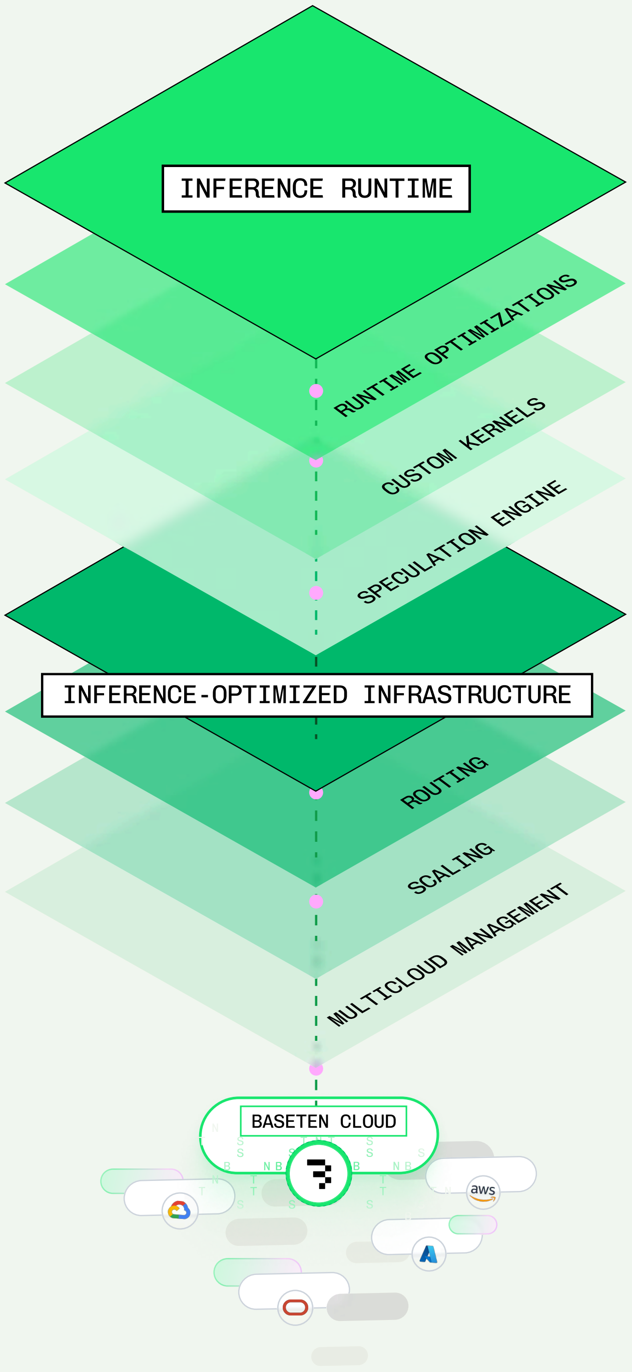
Deploying, optimizing, and scaling DeepSeek in production

To deploy, scale, and optimize the DeepSeek models for production, you'll need a combination of runtime and infrastructure optimizations to address these challenges and make them performant, reliable, and cost-efficient.

This is exactly what the Baseten Inference Stack is built for.

The Baseten Inference Stack abstracts away the complexity of deploying massive open-source models like DeepSeek by combining and packaging dozens of optimizations at the infrastructure and runtime layers. This enables teams to focus on their product and user experiences, instead of the inference optimizations needed to run powerful open-source models in production.

We'll break down the different tools and techniques the Baseten Inference Stack uses as an example of what it takes to run the DeepSeek models in production—performantly, reliably, and cost-efficiently at scale.



Deploying DeepSeek in production

We deploy DeepSeek V3 and R1 primarily on B200s to handle their 671B parameter MoE architecture while maintaining high model performance. We distribute DeepSeek's 256 experts across 8 GPUs (32 per GPU), and apply tensor parallelism to shard large matrix operations. This setup provides a performant foundation at the infrastructure level for DeepSeek deployments.

This is also the infrastructure foundation we use for our DeepSeek Model APIs, which run on B200 GPUs and offer on-demand DeepSeek V3 and R1 deployments (which consistently rank as the lowest latency and highest throughput DeepSeek models on OpenRouter, and include built-in support for streaming, structured outputs, and tool calling).

DeekSeek security and compliance

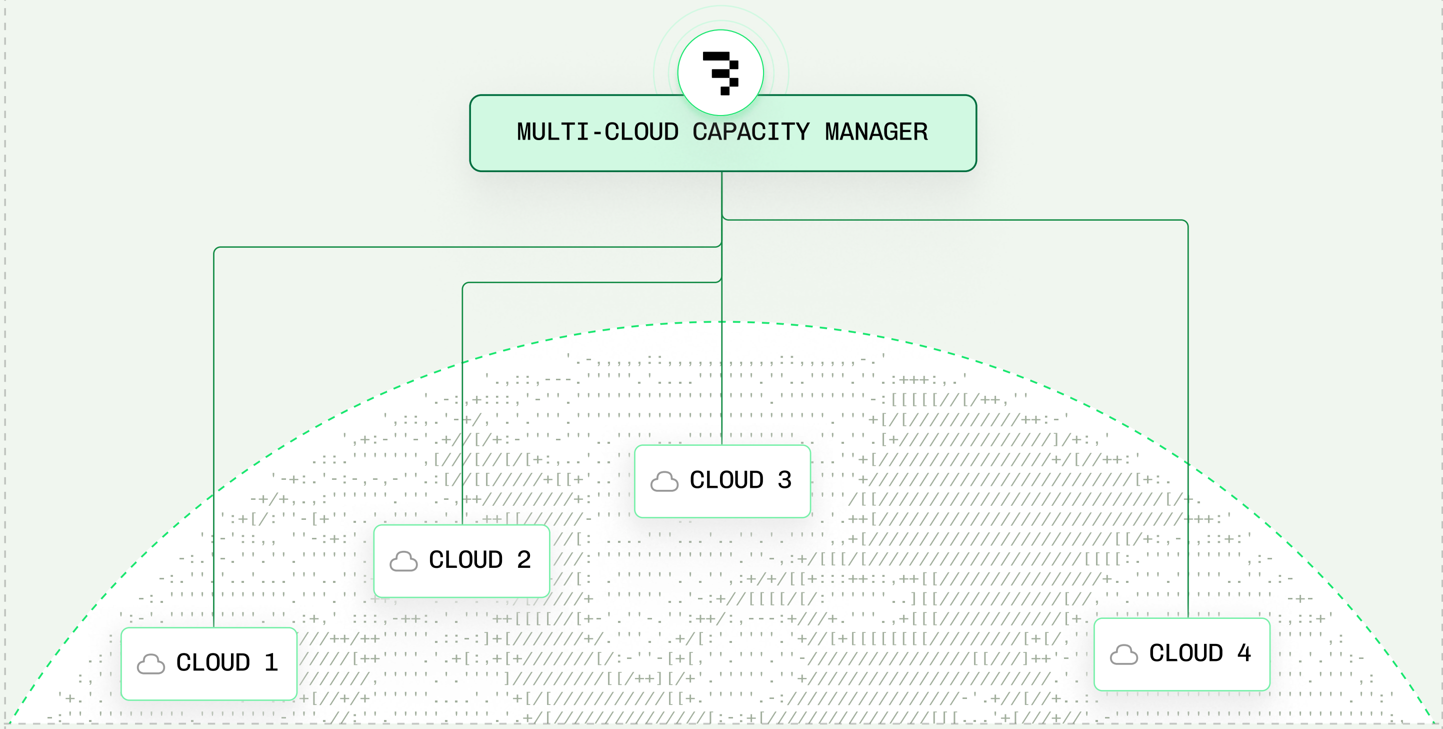
When DeepSeek V3 and R1 were first released, enterprises and startups were rushing to use them in production, only to face barriers in terms of security and compliance. To use DeepSeek securely in production—especially for sensitive industries, like healthcare and financial services—you need to know exactly where your data goes, and be able to control where it is and what it's used for.

That's why our DeepSeek Dedicated Deployments can be region-locked (e.g., so that they only run within the USA or EU). They can also be run in our cloud, which leverages compute from 9+ cloud providers (with SOC 2 Type II, HIPAA, and GDPR compliance by default), or can be self-hosted on your VPC (on any cloud provider) for full control.

Scaling DeepSeek in production

To scale DeepSeek effectively, we rely on a combination of Multi-cloud Capacity Management, intelligent request routing, SLA-aware autoscaling, and optimized cold starts.

Multi-cloud capacity management (MCM) is essential for scaling DeepSeek on B200 GPUs. We partner with 9+ cloud providers in dozens of regions globally to unlock almost unlimited capacity for our customers. That means that by using MCM, we can tap into a global supply of B200s to scale DeepSeek reliably across clouds (while respecting any region or cloud provider limitations).



Without MCM, you would have siloed resources and need to manually scale DeepSeek on different clouds and clusters when facing capacity constraints or hardware failures. Plus, you would need to make multi-year commitments to CSPs for always-on (and therefore overprovisioned) resources to accommodate traffic bursts or increasing scale.

CONTINUED >

Scaling DeepSeek in production

We use geo-aware load balancing to route requests to the nearest available replicas, lowering network round-trip time. Our geo-aware routing is also KV-cache-aware and LoRA-adapter-aware, meaning we prioritize sending similar requests to replicas with a warm KV state or the correct LoRA adapters already in memory. These routing techniques further improve end-to-end latency and time to first token (TTFT).

Finally, to meet SLAs reliably (i.e., to ensure high performance is consistent at scale), we custom-tune autoscaling settings per workload and optimize cold starts through various techniques. Without cold start optimizations, spinning up new replicas of large models can take up to an hour or more (imagine downloading nearly 700 GB of weights!). Fast cold starts let you spin up new model replicas quickly, minimizing latency spikes during traffic surges. Knowing that cold start times are fast also lets you overprovision less and scale down more aggressively, improving utilization (and therefore also cost).

These optimizations ensure we can horizontally scale DeepSeek with high availability and low variance in performance, even as traffic patterns evolve.

Optimizing DeepSeek in production

In addition to the foundational performance gains achieved through the infrastructure and routing optimizations outlined, we use a combination of model performance optimizations at the runtime level to optimize DeepSeek latency and throughput in production.

At the runtime level, we apply model quantization (FP4 for the weights and FP8 for the attention layer), speculative decoding (with multi-token prediction), KV cache reuse, and continuous batching. We've found TensorRT-LLM to be the most performant inference runtime, so we use it and its corresponding kernel optimizations. Our optimized DeepSeek model runtime also supports structured outputs out of the box.

All of these optimizations are part of our LLM model runtime, and the performance benefits they bring carry over to other model families, including Kimi, Llama, and Qwen.

Conclusion

The DeepSeek models are powerful tools for AI applications, from chatbots and assistants to RAG pipelines and agents. That said, achieving the performance, reliability, and cost-efficiency necessary for consumer-facing applications requires a lot more than just an inference framework and some GPUs.

DeepSeek's massive size makes it challenging to deploy, scale, and optimize for production. To do so, you'll need to leverage tools and techniques at the infrastructure and model runtime layers, like cross-cloud capacity management, tensor and expert parallelism, KV caching, and decoding techniques.

We package all of these optimizations in the Baseten Inference Stack as part of our DeepSeek LLM runtime, which powers dedicated deployments of DeepSeek as well as our production-ready Model APIs. If you want to use pre-optimized DeepSeek in your product or application, you can get started with our Model APIs, or talk to our engineers about a dedicated DeepSeek deployment that can autoscale to meet higher demand.

Contributors

A big shout out and thank you to our contributors:

Rachel Rapp

Tri Dao

Mahmoud Hassan