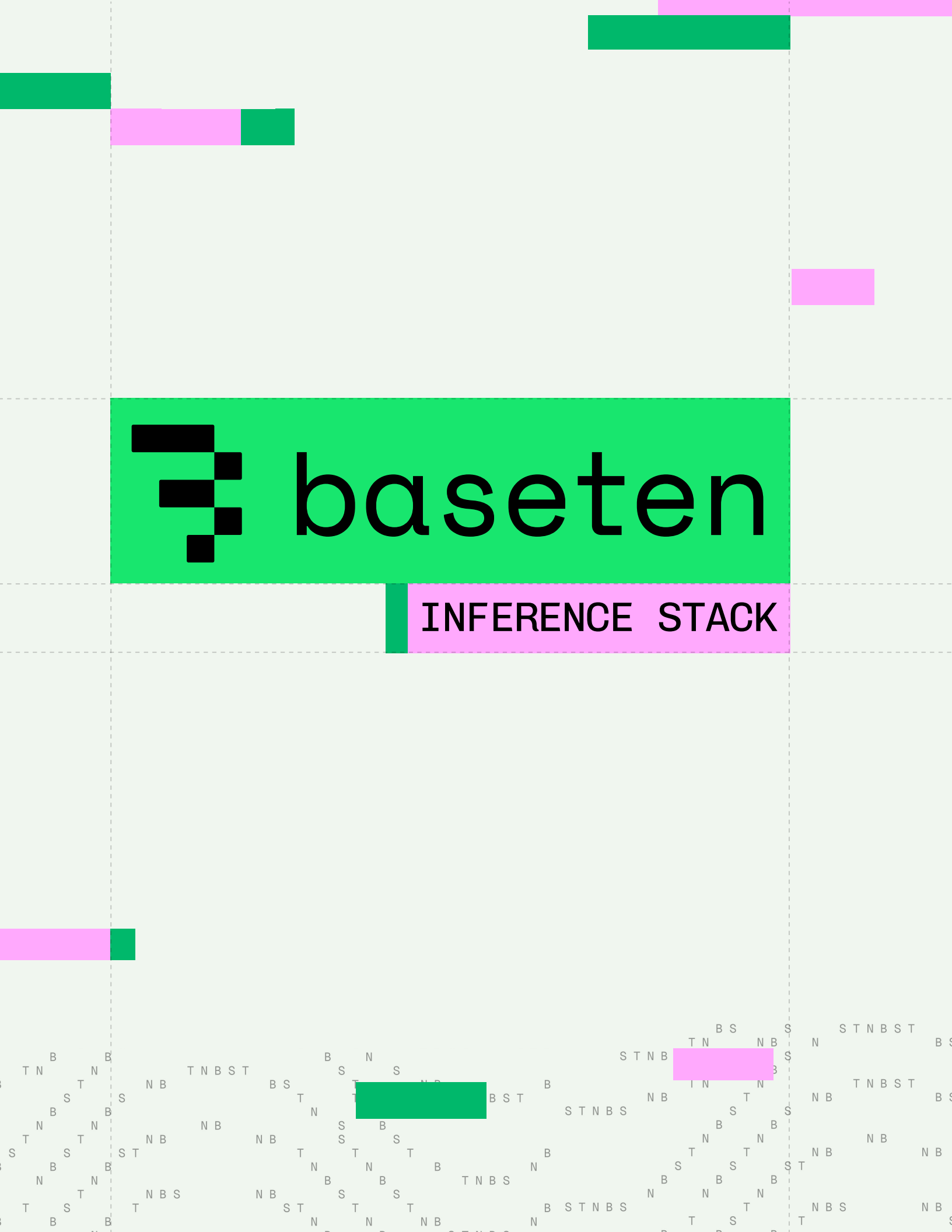

baseten
INFERENCE STACK

Table of Contents

The challenge of building fast, cost-efficient and reliable inference.

Running AI models in production isn't the same as building a demo. The shift to production adds strict requirements for latency, uptime, and cost. When those requirements aren't met, applications feel slow or unreliable, and ultimately, end-users suffer.

Spinning up a GPU with an inference framework (like vLLM or TensorRT-LLM) will get you decent performance but moving from decent to excellent is hard. Excellent means low, predictable latency under load, and at a price that makes sense at scale. Many systems stumble here.

Closing this gap takes calibration on every layer of inference, from the models, to the hardware to the many layers of software connecting them. The Baseten Inference Stack bundles those optimizations into a single platform, combining the best of open-source with our own proprietary enhancements. Every model or compound system you deploy on Baseten inherits these benefits by default.

In this guide, we'll cover:

1. The challenges that limit large-scale inference.
2. The infrastructure and runtime techniques that solve them.
3. How Baseten applies these techniques—plus modality-specific optimizations—in our Inference Stack.
4. How we enable cloud agnosticism to run the Baseten Inference Stack in our cloud, yours, or both.

Why building performant inference at scale is challenging

In production, performant inference means low latency, model performance, high throughput, and maximal reliability.

First, inference must be fast to ensure your product feels instant. Whether you're building code completion tools or dictation-driven applications, optimizing time to first byte and overall latency coupled with model performance directly impacts end-user experience.

But that's not enough. Inference must stay fast irrespective of the complexity of your workload, magnitude or variability of incoming traffic, and capacity (or reliability) of your cloud provider, all without compromising your economics.

New performance research is published at breakneck speed. Often, techniques are used in production that were introduced just weeks or months prior, like when DeepSeek V3 and R1 introduced Multi-head Latent Attention (MLA) to reduce KV cache demands on VRAM. When new models like DeepSeek drop, running them efficiently in production requires day-zero support for these techniques.

In short, it's nearly impossible to keep up with new techniques to support state-of-the-art model performance, in addition to reliability, and cost-efficiency without dedicated performance research and distributed infrastructure teams.

Challenge 1: Speed

There are many metrics used to measure inference speed across modalities: total request latency, time to first token (TTFT), time to first byte (TTFB), inter-token latency (ITL), and perceived tokens per second (TPS).

To make inference fast, you need to be strategic about regional workload placement to ensure that the servers are as close as possible to the caller and that you appropriately apply a number of techniques from recent model performance research, including but not limited to:

- Low-level optimizations like kernel fusion, memory hierarchy optimization, attention kernels, asynchronous compute, and PDL.
- Speculation strategies like draft-target speculative decoding, Medusa, and Eagle self-speculation.
- KV cache re-use and offloading to avoid recomputing large prefixes.
- Disaggregated serving to scale, prefill and decode on separate hardware with different runtimes.
- Post-training quantization in floating-point precisions with negligible gains to perplexity.
- Sending requests to geographically proximate GPUs and routing to warm KV and LoRA caches.

While each technique is powerful individually, it takes a combination to be viable in production as each optimization only targets part of the problem. For example, KV cache re-use improves TTFT tremendously but does little for inter-token latency.

Challenge 2: Reliability

High uptime (99.99% or better) is necessary but not sufficient for a reliable inference service. While avoiding downtime is essential, ensuring that your users always get access and see consistent performance is critical. In addition, the model outputs must consistently have low latencies as well. Many SLAs are based on p90 or p99 latency, or the idea that 90% or 99% of all requests are faster than a given target and it is not acceptable.

Achieving strong reliability metrics requires building robust infrastructure with:

- Autoscaling to increase capacity in response to traffic spikes.
- Active-active reliability and cross-cloud orchestration to fail over if nodes, zones, regions, or cloud providers go down.
- Strong alerting and automatic failure handling to mitigate hardware, networking, or driver failures from downstream vendors.
- Appropriate handling of multiple protocols (HTTPS/WebSocket/gRPC) to reliably support all modalities and traffic patterns.

Building this infrastructure requires rigorous and consistent abstractions to smooth over the numerous variations between cloud service providers.

Challenge 3: Cost-efficiency

If you had infinite resources, you could overprovision your systems to be safe when traffic increases. However, overprovisioning is enormously expensive and wasteful at scale. Utilization is the key metric to track to ensure sufficient headroom without unnecessary overprovisioning.

Inference is cost-efficient in production when you:

- Deploy models on hardware that fits their requirements.
- Scale different models independently as needed.
- Have access to enough resources to scale up confidently.
- Automatically scale back down as needed, without manual monitoring or intervention.
- Scale quickly (fast cold starts), even from zero, while keeping incoming requests safely queued.

A compounding challenge is that GPUs, especially in-demand recent-generation GPUs like B200s, must be acquired from cloud service providers via long-term reservations in blocks of nodes. Having the flexibility to leverage a hybrid infrastructure (proprietary cloud or Baseten) that gives the option to access additional compute resources when scale is needed can be a game changer in terms of cost optimization while maintaining high uptime and performance.

Key requirements of a production-ready inference service

To be production-ready, an inference service needs a unified approach to everything from the major pillars of infrastructure and runtime down to each millisecond-level implementation detail.

At Baseten, our Inference Stack is the cumulation of years of work on scalable, cloud-agnostic, fault-tolerant infrastructure paired with highly performant runtimes for every model and modality, wrapped in a cohesive developer experience. With the Baseten Inference Stack, you can:

- Build real-time AI applications with low-latency model inference.
- Access compute across multiple cloud providers for flexible capacity and economics.
- Maintain consistent performance across regions and clouds while retaining 99.99% uptime or better at scale.
- Take complete control of your inference infrastructure with file-based configuration and programmatic control—we don't believe in black boxes.

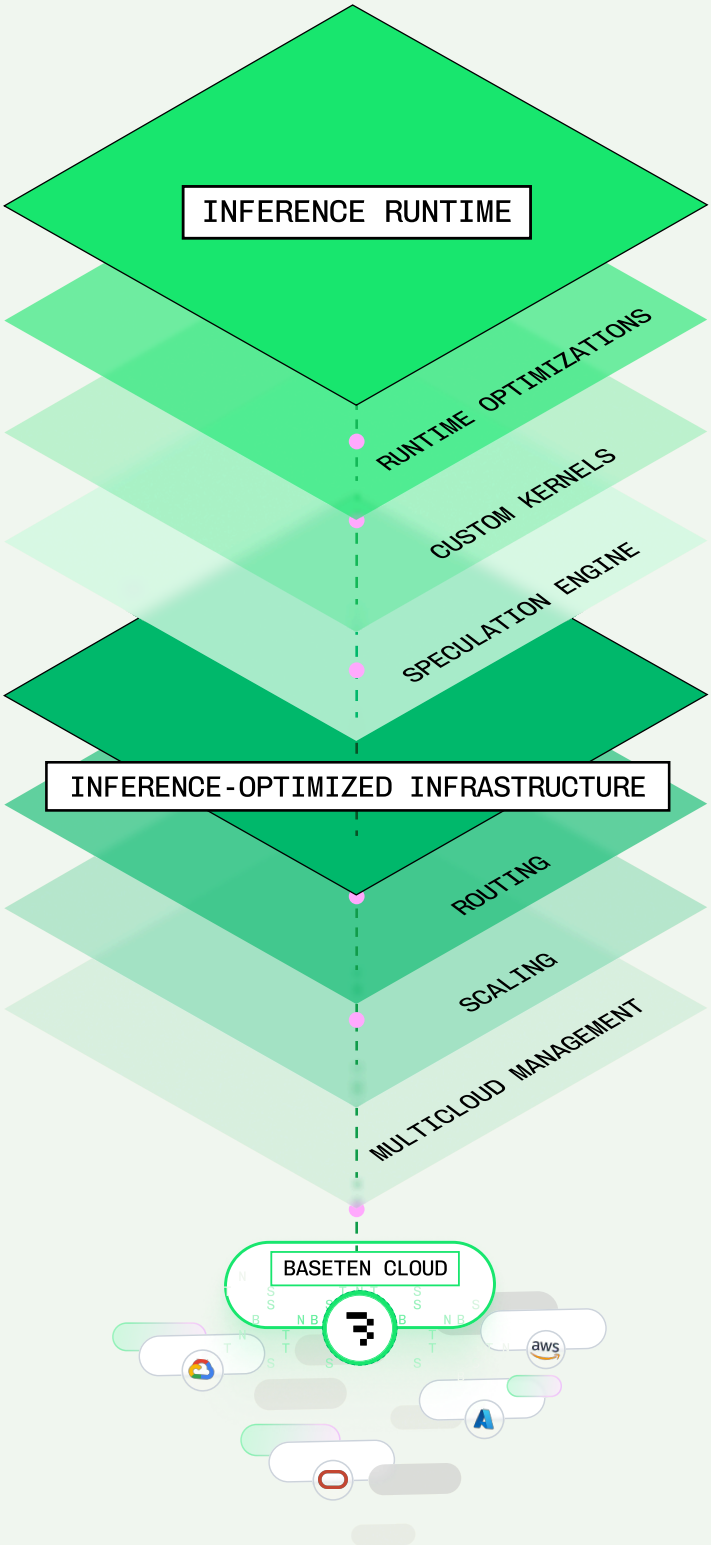
Our obsession with production shows up in every component of our Inference Stack, which powers everything on Baseten from our Model APIs to enterprise customers' self-hosted deployments while maintaining the highest standards of performance, reliability, and cost-effectiveness.

Baseten Inference Stack:
Combining infrastructure and runtime optimizations

The Baseten Inference Stack consists of two tightly integrated layers:

1. **Inference Runtime:**
   This layer focuses on how models actually run, including runtime optimizations, custom kernels, speculation engines, and more.

2. **Inference-optimized Infrastructure:**
   This layer includes request routing, autoscaling, and multi-cloud capacity management to ensure end users are reliably connected to the right resources.



Inference Runtime

Runtime optimizations sit on top of your inference infrastructure, and at the heart of our performance differentiation.

We use a combination of custom kernel optimizations, decoding strategies, modality-specific runtimes, and hand-picked features to squeeze every drop of performance out of models of every size and modality.

Custom kernels

AI model inference executes on GPUs, and kernels are the building blocks of GPU execution. Each kernel is a piece of low-level code that performs a specific computation. Poor kernel performance slows down the entire inference process, especially for high-throughput workloads.

We use custom kernel optimizations and select from state-of-the-art inference frameworks like TensorRT-LLM and SGLang to optimize specific models and workloads along with the hardware they run on. Specifically, we leverage techniques like:

- **Kernel fusion:**
  Reduces overhead by combining multiple operations (e.g., matrix multiplication, bias addition, activation functions) into a single GPU kernel. This minimizes memory reads/writes between operations and reduces kernel launch overhead.
- **Memory hierarchy optimization:**
  Ensures tensors are stored and accessed in the most efficient layer of GPU memory, prioritizing registers and shared memory over slower global memory.
- **Attention kernels:**
  Workload tailored attention kernels optimized to address speed, memory footprint, context length, and scalability.
- **Asynchronous compute and PDL:**
  Allows concurrent execution and uses kernel launch patterns supported by the recent Hopper and Blackwell architectures for better GPU utilization (relevant for workloads running on H100s and H200s).

CONTINUED >

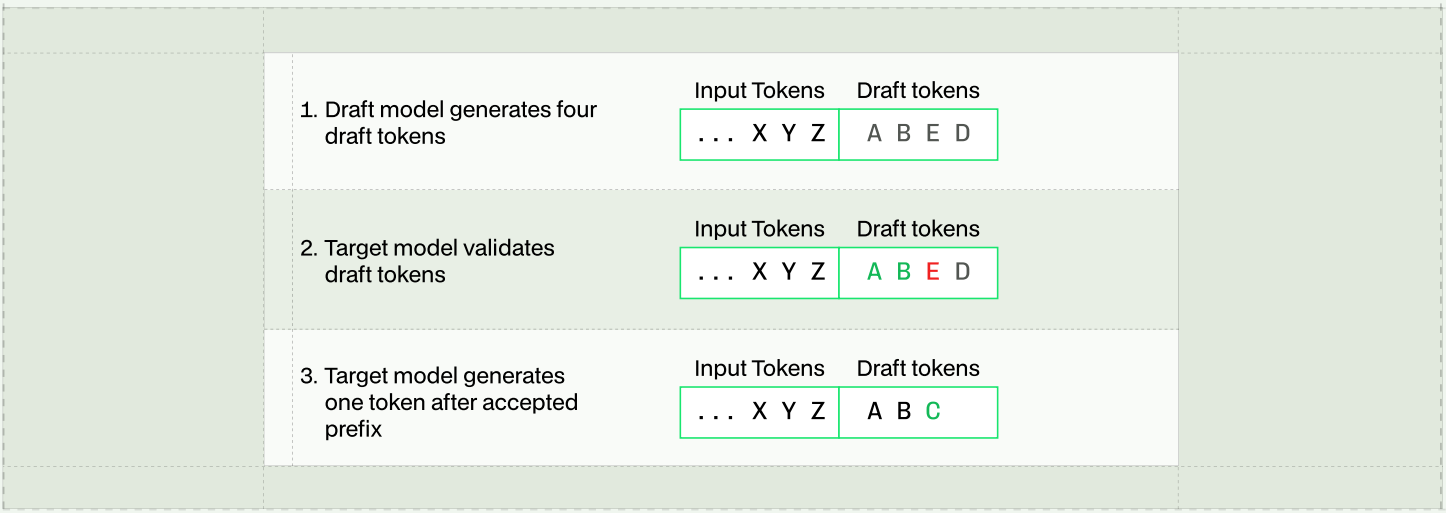
First, we benchmark frameworks like TensorRT-LLM, SGLang, and vLLM to select the best-performing framework based on workload characteristics and target hardware. Then, we build on top of what these frameworks provide, fixing bugs and extending functionality to support specialized inference workloads. Optimizing kernels gives us higher tokens per second (TPS), faster time to first token (TTFT), and improved performance numbers overall.

Speculation engine

The decode phase, where tokens are generated autoregressively, is often the slowest and most expensive part of inference. Decode generates tokens one-by-one and is generally bottlenecked on VRAM bandwidth.

Speculative decoding refers to a broad range of strategies around a common idea: what if the model could generate more than one token per forward pass. This would massively improve ITL and give users a better experience with lower total request latency and higher perceived TPS.

The classic approach to speculative decoding is a draft-target system. In this approach, a smaller draft model "guesses" several tokens ahead. The full (target) model then verifies these guesses. Because verification is cheaper than generation, this strategy can double or triple TPS if the draft model is good at guessing tokens.



CONTINUED ›

Speculative decoding performance in the wild varies massively by model, by topic, and by prompt. Our speculation engine supports numerous speculation strategies and can dynamically adjust parameters based on live traffic.

Supported speculative decoding methods include:

- **Eagle 3**
  Our most advanced draft model.
- **Lookahead decoding**
  Predicts next tokens based on previous context.
- **Medusa**
  A self-speculative strategy created by fine-tuning and grafting additional decode heads onto the target model.
- **Draft-based decoding**
  Implementations of draft-target speculative decoding optimized for specific use cases.

Speculative decoding shines at low batch sizes when there's spare GPU compute. At high batch sizes, it's dynamically turned off because verification becomes costly under compute saturation.

Optional quantization

Due to native training precision, by default most AI models run in FP16 or BFLOAT16, both 16-bit floating point number formats. In FP16, each parameter of model weights takes 2 bytes of VRAM to store and load, and the compute runs on Tensor Cores built for 16-bit numbers.

GPUs support other quantizations. Lovelace and Hopper GPU architectures introduced support for FP8, an 8-bit floating point format, while Blackwell added FP4, cutting the size in half again. These number formats offer a higher dynamic range than integer-based formats from previous generations.

Post-training quantization changes the precision of model weights, and optionally other values like KV caches. Quantized models require less VRAM to store, less VRAM bandwidth for decode, and offer higher FLOPS on Tensor Core compute for faster prefill.

CONTINUED >

However, quantization runs the risk of affecting model output quality. To mitigate this risk, we:

- Defaulting to floating point formats like FP8 and FP4 which preserve model quality thanks to their high dynamic range.
- Carefully selecting the optimal quantization scheme per model and GPU type.
- Benchmark models after quantization to ensure that perplexity and eval benchmark scores have only negligible change.
- Spot-check these benchmarks against real-world inference samples.

Model quantization on Baseten is always completely optional and transparent. We never quantize your models without permission and you retain full control over your model server configuration.

KV cache optimizations

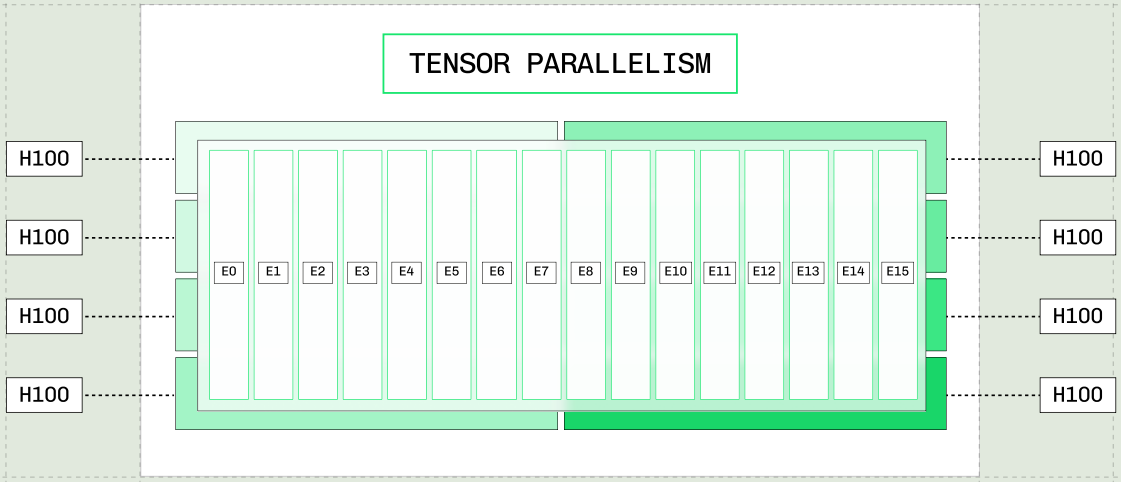
In LLMs, a lot of inference time is spent during the prefill phase—processing the prompt before generation begins. The key-value (KV) cache stores this prefill state, so it can be reused in subsequent requests with similar context, like a long system prompt or an entire codebase.

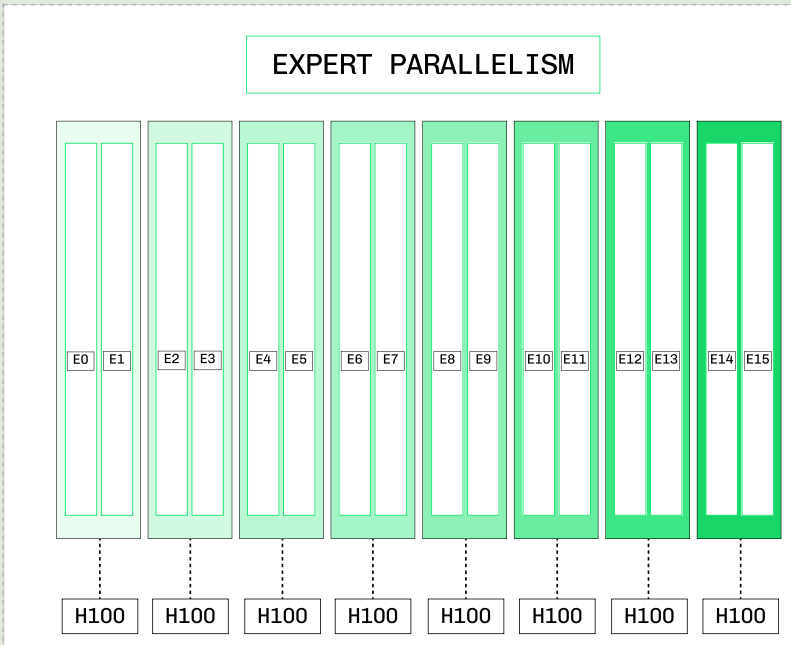
Baseten introduces several enhancements to improve the usefulness of KV caches:

- **KV cache-aware routing**
  Automatically routes requests to replicas that have the necessary context cached.
- **KV cache availability**
  Reuses cache for shared prompts like “You are a helpful assistant.”
- **CPU offloading**
  Moves unused cache blocks to CPU RAM to reduce GPU memory pressure.
- **Disk-based cache**
  Uses InfiniBand to distribute cache offloading across nodes—parallel disk reads perform nearly at CPU speeds.

Topology-aware parallelism

When serving large models on multiple GPUs and across nodes, model parallelism strategies like tensor parallelism (TP) and expert parallelism (EP) minimize communication overhead. Our runtime blends TP and EP along with other parallelism techniques to serve large models efficiently.





Request prioritization

For workloads that don't use disaggregated serving to separate prefill and decode onto different hardware, there's still a way to improve TTFT at the runtime layer: request prioritization.

Our runtime prioritizes prefill steps over decode, improving perceived speed with lower TTFT. As new requests come in, compute is allocated in favor of prefill. For example, if you have a request come in with 100,000 input tokens, prioritizing the compute cycles required for prefill ahead of decode steps from other requests in the batch ensures low TTFT (especially with KV cache reuse) while only minimally affecting ITL in bandwidth-constrained decode.

Continuous batching

Traditional batching waits for requests to arrive before processing them together. This introduces unacceptable latency, especially for real-time applications.
Our Inference Runtime uses the industry gold standard: continuous token-level batching. Instead of waiting for full requests, we batch at the token level.

This means:

- Requests can join an active batch mid-stream
- Latency is minimized without compromising throughput
- No idle time is spent waiting for request alignment

The result is higher GPU utilization and better average response times, especially under bursty traffic.

Structured outputs and tool use

Structured outputs, or returning LLM output based on a pre-determined schema, is essential for enterprise use cases. They enable programmatic interactions and composability with APIs, unlocking agentic workflows built on models that support tool use.

CONTINUED >

LLMs can be prompted to generate structured output, like JSON objects, but this is unreliable. Our Inference Runtime gives 100% guaranteed schema adherence for structured output by biasing logits according to a state machine generated prior to decode, ensuring no reduction in inter-token latency. This means your agents and tool calls always receive valid outputs from LLMs running on Baseten.

Modality-specific runtimes

When AI engineers talk about model performance, we often default to talking about LLM inference. From KV caching to speculative decoding, many of the most important techniques were developed with LLM inference in mind. However, there are many other modalities of generative AI models that can benefit from faster production inference.

The Baseten Inference Stack supports high-performance inference for every modality, including:

- Large language models like Llama, Qwen, and DeepSeek.
- Automatic Speech Recognition (ASR) models like Whisper.
- Speech synthesis models like Orpheus TTS.
- Embedding, reranking, classification, and reward models like Nomic Embed Code.
- Image and video generation models like FLUX.

One key insight in our modality-specific approach is that many non-LLM models, like TTS and embedding, have an LLM backbone. And some that don't, like Whisper for ASR, are still autoregressive transformers-based models which can be optimized with similar tools and techniques. Others, like diffusion-based image and video models, require entirely different approaches.

Large language models

Every aspect of the Baseten Inference Stack is built with LLMs in mind. LLMs are among the largest and most demanding of inference workloads, and benefit from nearly every technique and feature used in our Inference-optimized Infrastructure and Inference Runtime.

CONTINUED >

LLMs—often hundreds of gigabytes in size—benefit at the infrastructure layer from our cold start optimizations and active-active reliability. Once running, intelligent request routing improves KV cache and LoRA cache hit rate while balancing traffic across replicas. The largest models also need multi-node inference and disaggregated serving, both of which are provided by our infrastructure layer.

The runtime layer is designed to run the largest LLMs with low latency and high throughput for large scale deployments. KV cache re-use and request prioritization ensure fast TTFT for large inputs, while windowed attention extends that support to long context models. Quantization and our speculation engine provide low inter-token latency, keeping the user's perceived tokens per second high.

With support for every major LLM architecture and features like structured output and tool use, the Baseten Inference Stack is the best choice for LLM inference in production.

Speech-to-text models

Speech to text (aka automatic speech recognition) models are the foundation of transcription and dictation products as well as an essential component of voice pipelines.

The Baseten Inference Stack includes the world's fastest ASR runtime.

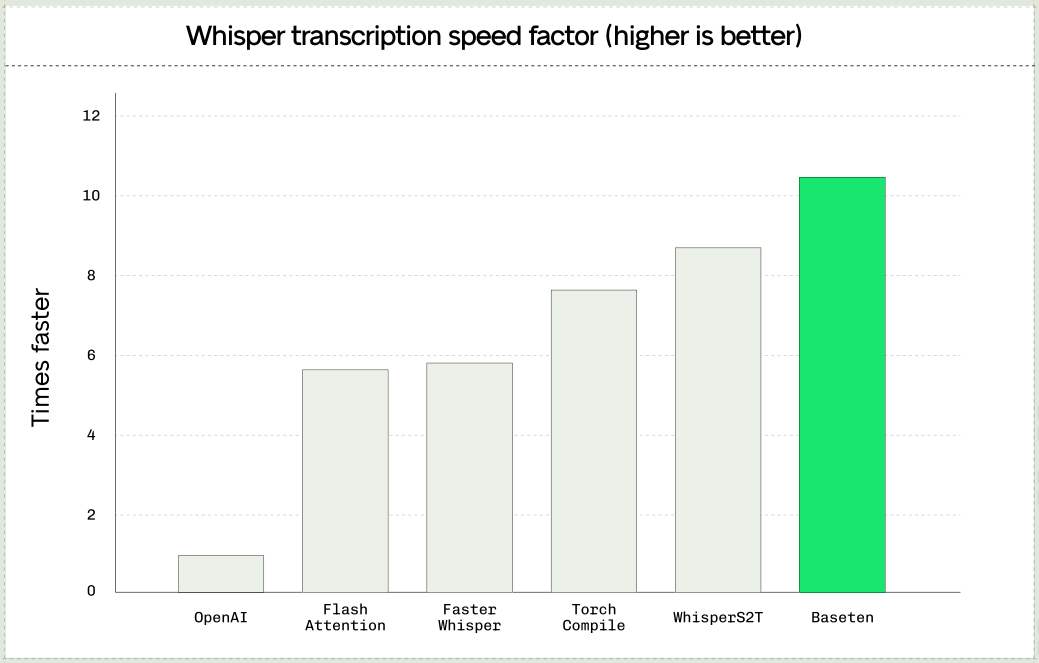
The main performance metrics for this modality are:

- Speed factor, or how many times faster than real time a transcript is generated. For Whisper, a powerful open-source ASR model, we achieve a speed factor of over 1,000.
- Round-trip latency, or how quickly a single chunk (up to 30 seconds) of audio can be transcribed and returned. For Whisper, we process these chunks in less than 200 milliseconds.
- Quality, as measured by WER (word error rate) for accurate transcripts that are usable in-product.

CONTINUED >

Speed factor is relevant for use cases like podcast transcription where users upload large files, while round-trip latency matters for real-time use cases like dictation.

We achieve state-of-the-art performance in both speed factor and round-trip latency using a combination of a TensorRT-LLM-based runtime and a proprietary chunking algorithm that supports retries for failed chunks, improving quality. Together, these techniques provide not only the fastest but also the most accurate ASR runtime as measured by word error rate

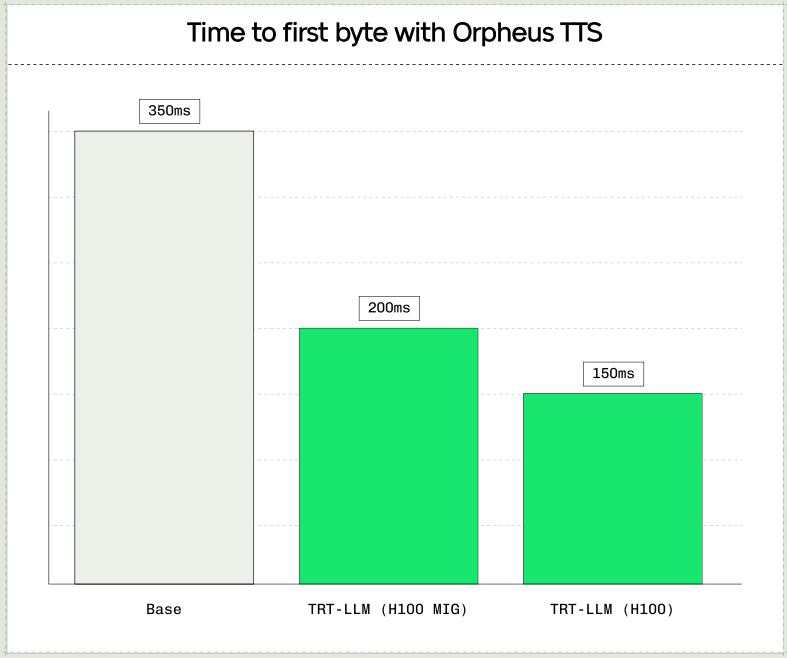


CONTINUED >

Text-to-speech models

On the other side of the speech pipeline, TTS (text-to-speech, aka speech synthesis) models are the voice behind voice agents. For TTS models, the main latency metric is TTFB, or the time it takes to generate the first byte of audio. This is important for online use cases like real-time conversation agents. The main throughput metric is simultaneous streams, or how many real-time users a single GPU can support (improving economics for production deployments).

Speech synthesis models like Orpheus TTS are built on an LLM backbone. By adapting our LLM tooling from within the Baseten Inference Stack and combining it with streaming protocols like WebSockets, we're able to achieve best in class latency and throughput for TTS models.



Thanks to this performance and throughput combined with Baseten's reliability, the foundation model lab behind Orpheus TTS selected Baseten as their preferred inference partner for running Orpheus in production.

Embedding models

Efficient embedding inference must support both high-throughput batch processing—such as vectorizing an entire corpus—and low-latency, single-query inference for real-time applications. That said, optimizing for one often degrades the other. Large token inputs risk out-of-memory (OOM) errors, small batch sizes leave GPU throughput untapped, and traditional runtimes struggle to balance performance across variable workloads.

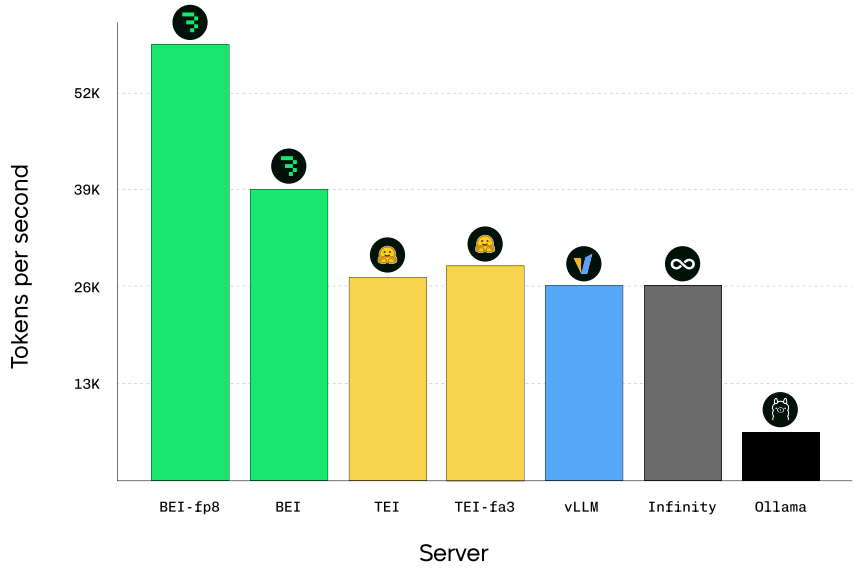
Baseten Embedding Inference (BEI) is built from the ground up to meet these demands. At the core is a TensorRT-LLM-based engine designed to balance latency and throughput by dynamically packing tokenized inputs based on sequence length rather than request count. This allows BEI to maximize GPU utilization without triggering out-of-memory errors, even under large or uneven workloads.

To further improve performance, BEI incorporates quantization to reduce memory requirements while maintaining model accuracy. It also uses fused transformer layers and optimized attention kernels to minimize memory bandwidth usage and reduce compute overhead. These optimizations make BEI well-suited for both batch-heavy ingestion workloads and millisecond-scale interactive applications.
At the infrastructure layer, BEI benefits from autoscaling to process large corpora of documents in parallel, along with strong queueing primitives that ensure requests aren't lost while scaling.

As a result, BEI is the most performant embeddings solution on the market: over 2x higher throughput than the previous-leading solution, with 10% lower latency. BEI supports real-time applications like RAG and search, while also scaling up to process entire corpora efficiently. Whether you're embedding documents or reranking outputs, BEI delivers the performance needed to systems fast and cost-effective.

CONTINUED >

Mistral-7B — Vector DB Throughput



Mistral-7B — Idle Latency (ms)

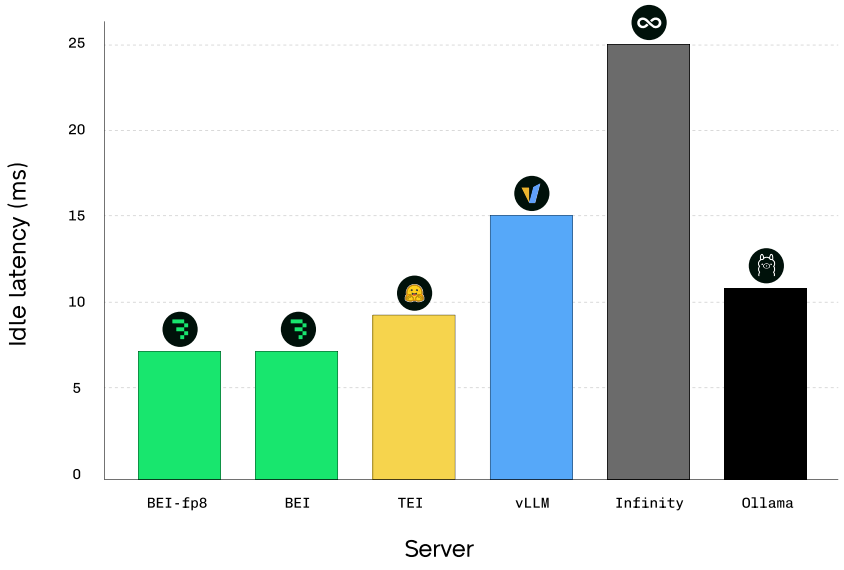


Image and video generation models

Image and video generation models require a very different runtime than other modalities as the runtime must support diffusors. For these models, the main performance metric is end-to-end request latency to generate an image or video.

Diffusors requires its own set of custom kernels, from torch compilation and efficient attention to post-training quantization. Most of the gains in image model inference come from this low-level CUDA work.

However, one unique thing about image and video models is that they have much more latitude than LLMs and similar models in trading off small quality losses for big speed gains. Techniques like latent consistency for few-step inference, along with more traditional approaches like distillation, can yield major performance gains. And there are plenty of prompt-level adjustments, from image resolution to diffusion step count, to dynamically trade off between speed and quality.

Baseten's Inference Runtime supports compilation and the latest kernel optimizations for image and video models across both popular open-source inference frameworks and custom servers.

Inference-optimized Infrastructure

Runtime optimizations matter when your model executes; your infrastructure gets you to that starting line. With every hop, there is room for latency and failures to slip into your system.

Intelligent request routing

Most load balancers assume that any request can be handled on any machine with the same performance. Baseten's load balancers go a step further by relying on KV caches and model adapters to achieve both latency and quality goals.

Serving requests on hardware that is physically closer to the requests' origin, or already has useful information loaded into memory, can make a huge difference in inference speed. To route requests intelligently, we leverage techniques like geo-aware load balancing with KV cache-aware and LoRA-aware routing.

Geo-aware load balancing

Geo-aware load balancing is critical for latency-sensitive workloads that are globally distributed. It reduces network latency by sending requests to the most proximate compute. Otherwise, data has to travel longer physical distances through multiple network hops, significantly increasing transmission time and the chance of congestion or packet loss.

For instance, if a request comes to your model from Germany, and you have model replicas running in US East, US West, and London, geo-aware load balancing would send the request to London. This is especially important for real-time workloads, like AI phone calling and agents.

KV cache-aware routing

The Key-Value (KV) cache stores key and value tensors from the attention layers of a transformer during inference. Instead of recomputing past context for each new token, the model reuses this cached data to generate the next token faster. It's essential for low-latency text generation for model families like Llama, DeepSeek, and Qwen.

KV cache-aware routing involves sending requests to replicas that already hold relevant context, like the cached prompt. For instance, if you have 10 different replicas, sending your requests to the replica that has recently seen similar requests increases the chances of a cache hit.

This can dramatically reduce prefill latency, especially for chatbot-like workloads. For example, the latest messages (requests) would ideally be sent to the same replica that processed the previous requests. Otherwise, the new replica will also have to load the missing context into memory.

LoRA-aware routing

Similar to KV cache-aware routing, LoRA-aware routing involves sending requests to replicas that have a particular LoRA (low-rank adaptation, similar to a fine-tuned model head) loaded into memory.

LoRA-aware routing is often relevant for image generation and LLM use cases, which can leverage hundreds to thousands of LoRAs in production. Sometimes, in these cases, only a subset of the LoRAs can fit into memory; LoRA-aware routing ensures each request is directed to a replica with the correct adapter pre-loaded. Otherwise, the replica has to first download the relevant LoRA to serve the request.

Protocol flexibility

Different protocols are better suited for different workloads. In addition to the typical HTTP request-response paradigm, we leverage WebSockets and gRPC support for workloads on a case-by-case basis to improve performance.

CONTINUED >

In contrast to HTTP—which has the additional overhead of opening and closing a new connection per request—WebSockets enable a continuous client-server connection. WebSockets are useful for transmitting unstructured and real-time data (like audio), where the server receiving the request can parse it and process it downstream. They're particularly relevant for use cases requiring real-time streaming, like AI phone calls.

Similar to WebSockets, gRPC also enables bi-directional streaming support, but for structured data. Requests transmitted via gRPC should follow a predefined schema, which takes away the load of having to parse the input. This additional validation layer makes gRPC slightly slower than WebSockets, but is especially useful for use cases that involve translating mediums across a request (like going from text to video).

Autoscaling

The importance of performant autoscaling cannot be overstated. It determines:

1. How quickly you can scale to meet a burst in traffic
2. How cost-efficient your hardware usage is when traffic slows down

Whether spinning up a new model replica takes five minutes or 30 seconds makes a huge difference in terms of your P99 and P90 latency (and, of course, your end-user experience). We use a combination of in-house cold start optimizations and expert tuning to set parameters for optimal performance with awareness of customers' target SLAs.

SLA-aware autoscaling

Hitting latency and cost-effectiveness targets requires configuring your autoscaling settings accordingly. That said, determining these parameters often requires implicit knowledge about how they suit different types of workloads.

Settings you have to configure include the:

- Minimum and maximum number of replicas (the lowest and highest number of active replicas)
- Autoscaling window (the time window for traffic analysis before replicas are scaled up or down)
- Scale down delay (waiting period before unused replicas are removed)
- Concurrency target (number of requests a replica should handle before scaling)

CONTINUED >

Within these boundaries, our autoscaler is optimized to dynamically adjust the number of active replicas to handle variable traffic and meet your SLAs (while minimizing idle compute costs).

Baseten provides guidance on the right parameters for specific workloads and programmatically exposes all autoscaling settings so power users have full control to tune as they see fit.

Fast cold starts

A "cold start" is the entire time it takes to provision GPU resources, load model weights, images, and other dependencies onto the GPU, activate the model on the GPU, and begin serving traffic. During this process, you also need to hold excess requests in a queue and release them as replicas become available.

With new GPUs like B200s offering much larger VRAM, models are getting larger in turn. DeepSeek-R1 weighed in at 671B parameters—that's 720 GB of model weights in FP8—and Llama 4 Behemoth promising to top 2 trillion parameters, cold start optimizations are more important than ever. Without substantial engineering effort at each stage of the process, cold starts for these incredibly large models can take hours.

Instead, we want to measure cold starts on the scale of seconds. 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 thus cost.

To optimize cold start times, we use a combination of:

- Network acceleration via parallelized byte-range downloads.
- Specialized pods that accelerate loading times.
- Caching of model weights, builds, dependencies, and data at both the cluster and node level.

As a result, we're able to cold start most models in seconds and the largest models in a handful of minutes.

Independent component scaling

AI applications are increasingly built on multi-model, multi-stage compound AI workloads where several inference steps need to be run in a coordinated fashion. If you have two or more models, they are likely to have different hardware and scaling needs.

In this situation, provisioning resources as a monolith results in either bottlenecks or overprovisioning. Instead, we decomposed autoscaling to scale individual processes independently. This enables us to:

- Right-size resources per step, running each AI model on best-fit GPUs and business logic on cheap CPU instances.
- Remove processing bottlenecks by enabling independent replica counts for each step in a pipeline.
- Ensure cost efficiency by avoiding overprovisioning steps that have lower demand.

For instance, when running a transcription workload, you can separate audio chunking (a processing step) from the actual transcription (which uses an AI model). The first step can be run on a less-expensive CPU, and the second step can scale independently to accommodate the larger number of audio chunks. This lowers costs by:

- Preventing idle GPUs, since chunking is run in parallel, preventing bottlenecks.
- Using the correct hardware per step (i.e., using more powerful GPUs only for the steps requiring them).

We built an entire SDK and toolkit for ultra-low-latency compound AI systems on top of this design: Baseten Chains. Chains is what enabled us to build the fastest, most cost-efficient Whisper transcription on the market.

Disaggregated prefill and decode phases for LLMs

LLM inference has two phases:

- **Prefill phase:**
  Loads the context (including the full input prompt and communication history) and builds an initial Key-Value (KV) cache.
- **Decode phase:**
  Generates the response token-by-token, using the KV cache for faster token prediction.

Prefill and decode have different resource needs (prefill is generally compute-bound while decode is generally bandwidth-bound) and benefit from different kernel optimizations.

Disaggregating prefill and decode by running each phase on separate GPUs reduces latency through independent scaling. For instance, you can assign more GPUs to prefill to reduce TTFT. And inference is overall faster thanks to runtime enhancements from phase-specific optimization as well as the elimination of competition for resources between the two phases.

Multi-cloud capacity management

Multi-cloud capacity management is a set of automations, tools, and practices around provisioning and operating compute resources across multiple cloud service providers (CSPs) and regions in a standardized and repeatable manner.

Baseten's multi-cloud capacity management efforts:

- Ensures high uptime (99.99%) through active-active reliability
- Enables the lowest possible latency through flexible compute allocation
- Supports data residency and sovereignty requirements
- Unlocks an optimal customer cost-performance ratio

Our north star is to deliver consistent performance regardless of CSP, region, or unique customer requirements.

CONTINUED >

Active-active reliability and GPU failover

Active-active reliability refers to a system architecture where multiple replicas or regions are live and serving traffic simultaneously. If one fails or degrades, the others continue handling requests without disruption, ensuring high availability and fault tolerance via seamless failover.

Active-active reliability improves both uptime and economics by:

- Insulating against provider, region, and node-level outages (e.g. if AWS us-east-1 goes down, your service stays up)
- Balancing workloads across compute to provide the right cost profiles and SLA guarantees.
- Gaining access to more compute by spreading your workloads across multiple providers.

Baseten's Inference-optimized Infrastructure handles the key challenge of active-active reliability—determining optimal workload placement timing and location—to ensure that you get all of these benefits for every deployment.

Cross-cloud and multi-cluster orchestration

Cross-cloud orchestration is the "how" behind active-active reliability, GPU failover, and multi-cloud scaling.

Broadly, it's the practice of building consistent abstractions across CSPs so that resource provisioning and allocation is identical from provider to provider. Each CSP has its own unique wrinkles, from its networking stack to its exact resource SKUs for GPUs and associated CPU, RAM, and storage. The ultimate goal in cross-cloud orchestration is to be able to treat a given GPU from any CSP as a fungible commodity while retaining high uptime SLAs.

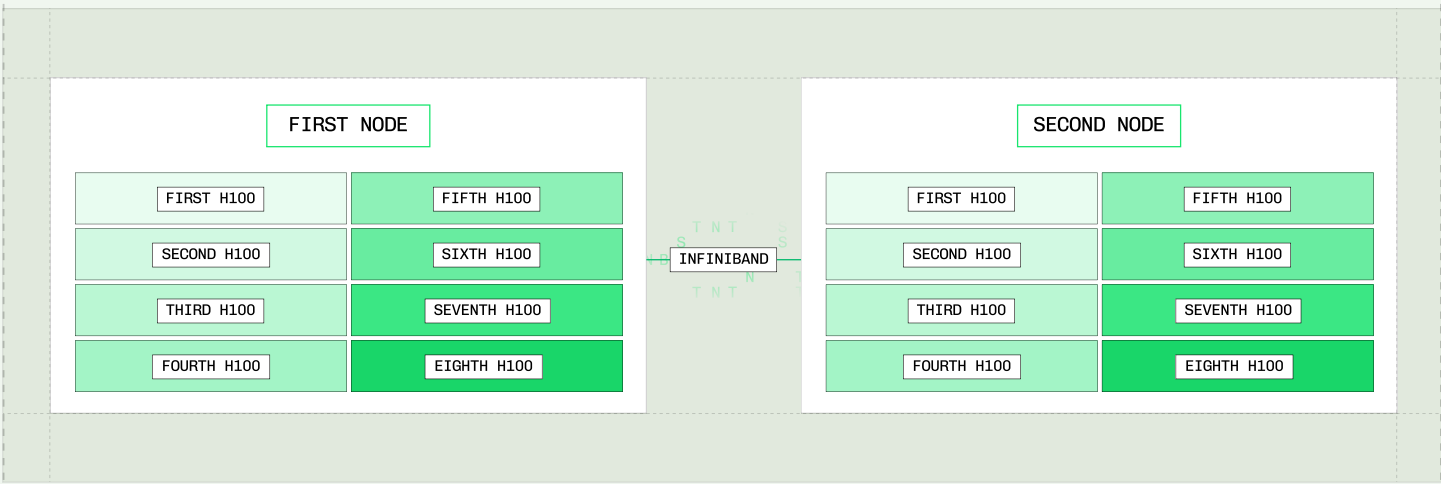
CONTINUED >

Cross-cloud orchestration ensures global resilience, protecting against provider-level and regional failures. It also provides greater access to compute, especially in-demand resources like B200 GPUs, as you can buy compute from anywhere.

Baseten's Inference-optimized Infrastructure supports latency-sensitive workloads and workloads with regulatory or data sovereignty constraints across any region or cloud provider thanks to our cross-cloud capabilities.

Multi-node inference

Every infrastructure challenge is built around the assumption that models run on up to eight GPUs, or a single node. However, some models are so large that they need more than eight GPUs to run efficiently. For these cases, our Inference Stack supports multi-node inference.



Multi-node adds two challenges at the infrastructure layer:

- Provisioning and lifecycle management for multiple, physically interconnected GPU nodes adds complexity to capacity management.
- Different cloud providers use different interconnect and networking technologies, from NVIDIA's Infiniband to custom solutions, with varying bandwidths.

Baseten's Inference-optimized Infrastructure abstracts away these challenges and enables serving extremely large models on multi-node resources.

Security and compliance for inference

For AI models to power mission-critical applications, they need to be secure and compliant on top of being fast and reliable. That's why security and compliance are baked into every layer of the Baseten Inference Stack.

Companies around the world trust us with their most sensitive workloads. We're equipped to meet the unique compliance needs of highly regulated industries, and we maintain compliance with SOC 2 Type II, HIPAA, and GDPR.

Baseten's Inference-optimized Infrastructure helps developers build compliant services. Our geography-based routing enables compliance with data sovereignty laws, and all requests are sent directly to workload planes without going through any intermediary control plane.

We're in the business of providing AI inference infrastructure, not using customer data. By default, we never store model inputs, outputs, or weights (if a user chooses to send async requests, the associated inputs are temporarily stored for up to 24 hours in a secure queue and requests are permanently deleted after inference occurs). Caching weights is optional, and they can be permanently erased at any time.

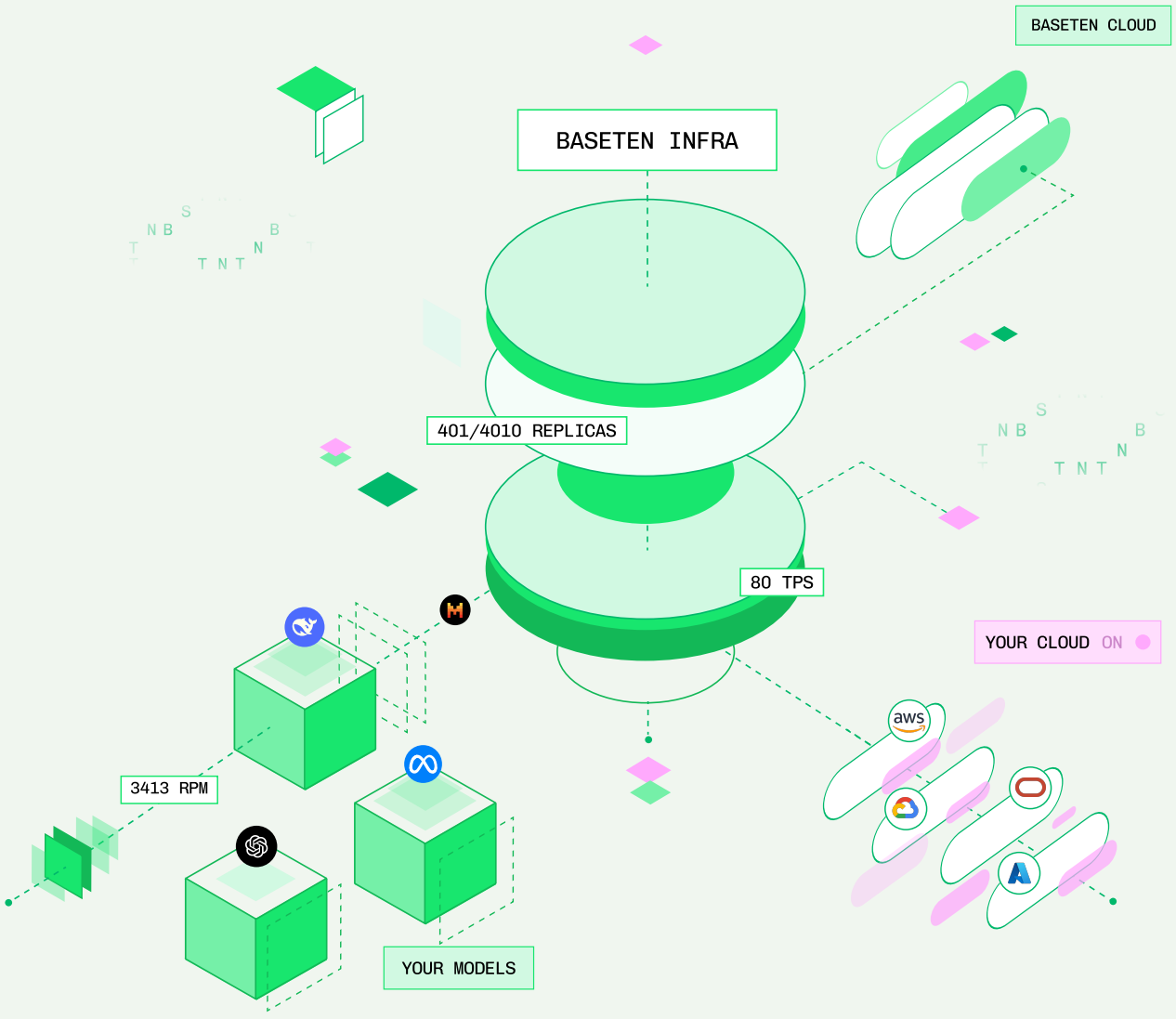
Additionally, we implement robust security features to protect your data and workloads, including data encryption, container security, network and access controls, workload isolation, and extensive penetration testing. We ensure the necessary isolation to protect your workloads with container security, network security, and strict privilege management.

For enterprises with even more extensive security and compliance needs, we offer single-tenant environments and self-hosted deployments that provide enhanced control over regulatory adherence, data localization, and customization of their inference infrastructure. We discuss hosting options in more detail below.

```
Note: If the user chooses to use async, then the model's output is never stored
but the input is temporarily stored for async function to work, and then it's
permanently deleted.
```

The Baseten Inference Stack in our cloud or yours

We built Baseten from day one to be Kubernetes-native and cloud-agnostic. This philosophy has made it possible for us to go multi-cloud early.

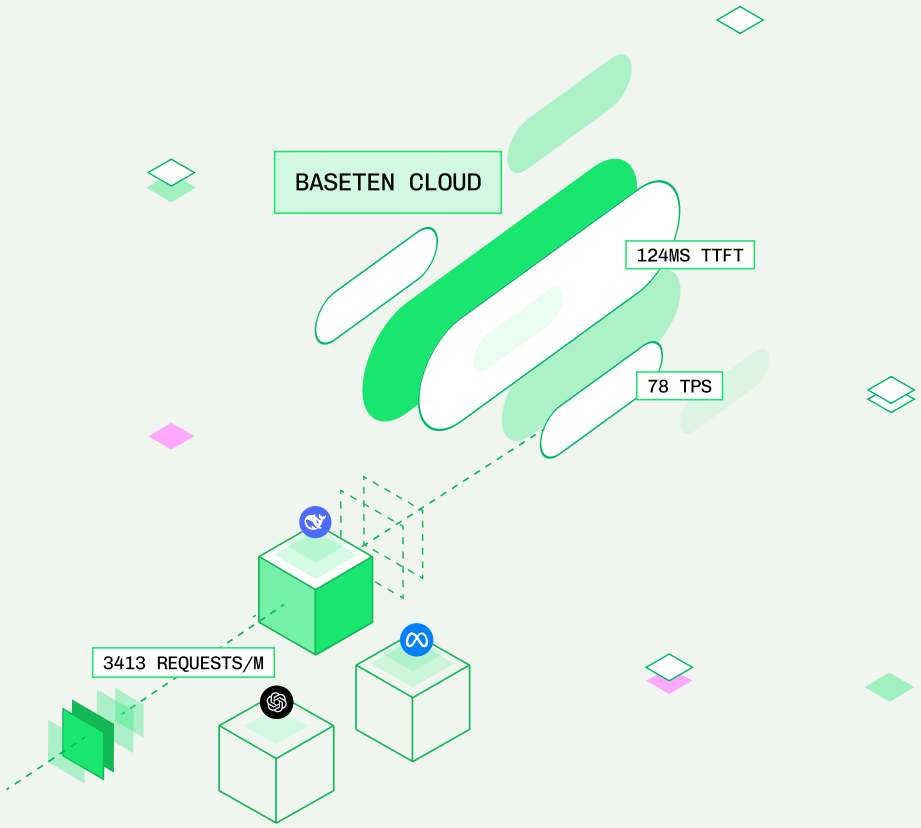


Baseten Cloud

Baseten Cloud offers fully-managed inference infrastructure with all the performance benefits of the Baseten Inference Stack. Today, we sit across 7+ public clouds and are constantly adding new regions and capacity. Unlike single-cloud solutions, Baseten Cloud is architected to run across providers and regions, giving the flexibility to avoid vendor lock-in while optimizing for latency, GPU availability, and cost.

The flexibility in our infrastructure lets us offer our customers flexibility in where they run their workloads. Baseten can deploy workloads supported by our entire Inference Stack into your VPC, and optionally spill over excess traffic back to infrastructure in a Hybrid deployment. This standardized and systematic cross-cloud capability is unique among inference providers.

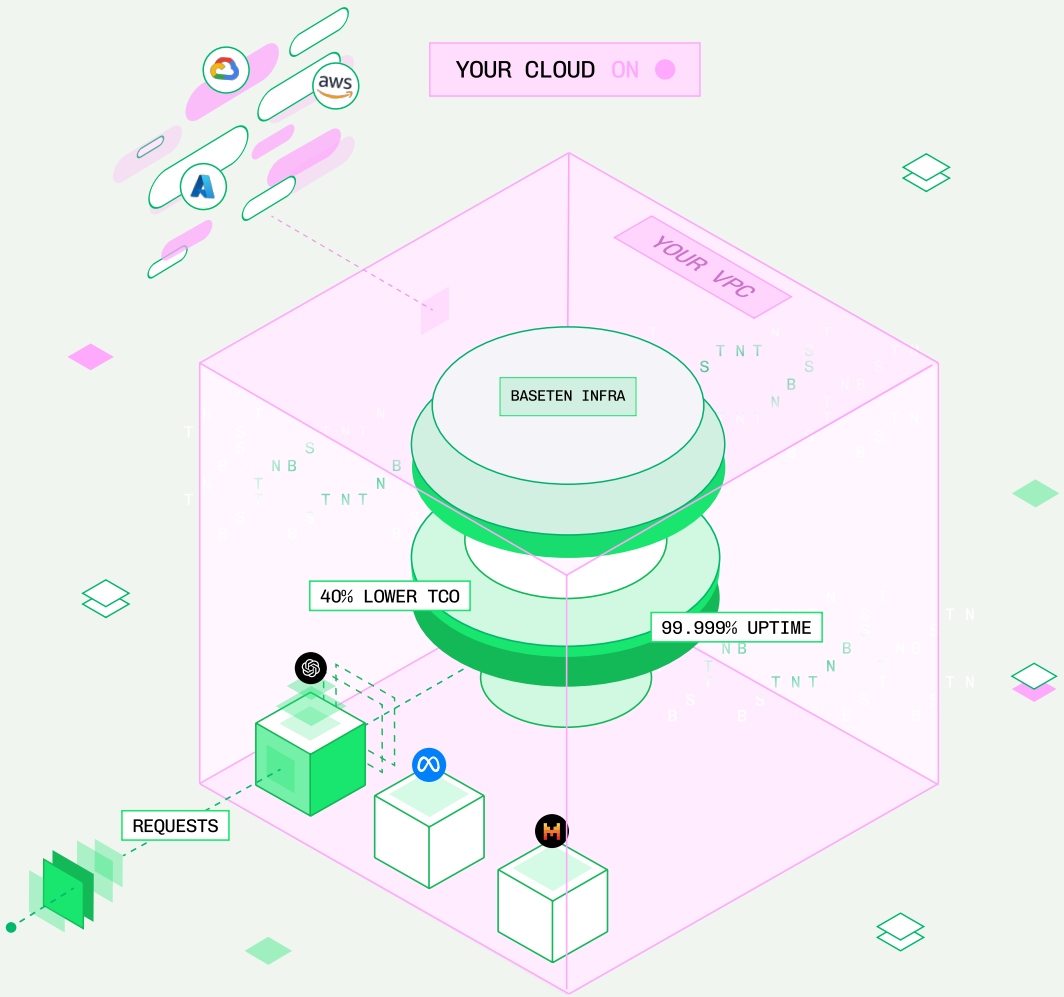
Basten Cloud is designed to meet the needs of teams scaling AI-native products, with massive cross-cloud autoscaling, global compute availability, and the flexibility to region-lock workloads as needed for compliance. We're SOC 2 Type II certified, HIPAA and GDPR compliant, and we never store model inputs or outputs.



Baseten Self-hosted

For teams with strict data security, privacy, or infrastructure requirements, Baseten Self-hosted provides full access to our platform within your own cloud environment. You get all the advantages of the Baseten Inference Stack—kernel and runtime performance, routing, autoscaling, and observability—while maintaining complete control over your data, compute, and networking.

Self-hosting ensures that no data ever leaves your environment. You know exactly where your data is processed, how it flows, and who has access. Inputs and outputs are never stored or shared, offering peace of mind for teams working with sensitive IP or regulated workloads.



Baseten Hybrid

While self-hosting can be necessary for many teams with data residency or compliance requirements, self-hosting alone often can't accommodate traffic spikes or sudden scale. The result is a trade-off between control and flexibility that's hard to manage without over-provisioning.

Baseten Hybrid removes this constraint by combining self-hosted control with optional, elastic spillover to Baseten Cloud. You define where your workloads run—whether entirely in your cloud or with dynamic routing to Baseten Cloud when demand spikes. It's the same Baseten experience, with no engineering effort required to manage infrastructure, compliance, or scaling logic.

Baseten Hybrid is especially appealing to customers with large pre-existing commitments or favorable GPU allocation and pricing from cloud providers, letting them spend down existing commits while accessing flex capacity as needed. Whether you're meeting compliance requirements or managing cost-efficient scale, Baseten Hybrid delivers cloud elasticity with the control of self-hosting.

The future of high-performance AI inference

Consistently achieving demanding targets for latency, model performance, uptime, and cost in production requires a holistic view of inference across applied performance research and distributed infrastructure.

The Baseten Inference Stack combines our Inference Runtime based on years of model performance optimizations on top of our Inference-optimized Infrastructure that we built from day one for scale, flexibility, and security. It powers every model of every modality deployed on Baseten.

Achieving similar performance and reliability from scratch requires a team of specialized engineers working full-time to apply solutions like request routing, autoscaling, and capacity management alongside runtime optimizations, custom kernels, and speculation engines across regions and cloud providers.

The Baseten Inference Stack gives you all of these capabilities out of the box, while leaving you with full visibility and control over how your model inference and infrastructure is configured.

Most importantly, we continue to evolve our stack with day-zero support for new models, improved runtimes for each generation of GPU hardware, the latest performance research techniques, and new AI engineering patterns like compound AI. These constant improvements keep your models current with the state of the art so that you never fall behind despite the pace of the AI industry.

If your AI product is starting to hit a scale that exposes the limitations of off-the-shelf models, other inference providers, or homegrown solutions, talk to our engineers about how Baseten can support your models in production.

Contributors

Contributors are listed in alphabetical order by last name.

Ke Bao, Bryce Dubayah, Michael Feil, Pankaj Gupta, Mahmoud Hassan, Matt Howard, Phil Howes, Philip Kiely, William Lau, Zhang Lu, Colin McGrath, Rachel Rapp, Abu Qader, Ujjwal Sarin, Phillippe Siclait, Helen Yang, Bryan Zhang, Yineng Zhang