Red Hat

Get started with AI inference

Accelerate your path to efficiency

Table of contents

Introduction 3

Key terms at a glance 4

The evolution of large language models 7

Challenges of inference serving 9

A full-stack approach to inference performance 10

A dual approach to model efficiency 12

1: Optimizing the inference runtime (vLLM) 12

2: Optimizing the AI model 14

Red Hat AI 18

What is Red Hat AI? 18

Optimizing inference with Red Hat 20

Next steps 22

Introduction

Optimizing AI model inference is among the most effective ways to cut infrastructure costs, reduce latency, and improve throughput, especially as organizations deploy large models in production.

This e-book introduces the fundamentals of inference performance engineering and model optimization, with a focus on quantization, sparsity, and other techniques that help reduce compute and memory requirements, as well as runtime systems like Virtual Large Language Model (vLLM), which offer benefits for efficient inference.

It also outlines the advantages of using Red Hat's open approach, validated model repository , and tools such as the LLM Compressor and Red Hat® AI Inference. Whether you're running on graphics processor units (GPUs), Tensor Processing Units (TPUs), or other accelerators, this guide offers practical insight to help you build smarter, more efficient AI inference systems.

Key terms at a glance

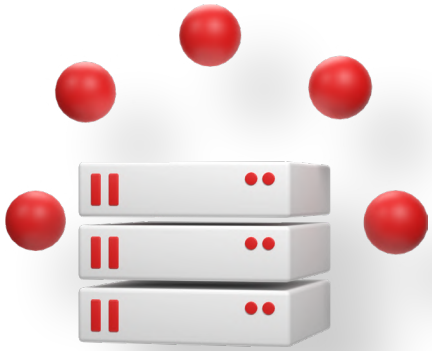
Understanding model components

Activations

Activations are temporary data generated as a model processes information (input tokens), similar to intermediate results produced during a calculation. They typically require high precision for accurate results.

Weights

Weights are the learned parameters or settings of an AI model, much like configuration files or settings in traditional software. They determine how the model analyzes and predicts data and can often function effectively at reduced precision.



Quantization

Quantization reduces the size and resource requirements of AI models by storing their parameters (weights) and intermediate data (activations) in lower precision formats, using fewer bits per value. This technique helps manage resources efficiently, similar to compressing files on a computer. Done correctly, it **does not significantly degrade the performance of the model**.

- **Weight quantization** reduces the storage size of the model's parameters, **allowing more efficient use of memory during inference**.[1]
- **Activation quantization** minimizes the memory requirements of intermediate outputs (temporary data) during inference, **making execution faster and more efficient**.[2]
- **KV cache quantization** shrinks the memory footprint of cached key value tensors, helping models **handle long prompts and concurrent requests more efficiently**.[3]

Precision levels at 16-bit, 8-bit, and 4-bit quantization:

- **16-bit (FP16/BF16)** is standard precision, preserving accuracy but demanding significant memory, making it costly for very large models.
- **8-bit (FP8/INT8)** reduces memory usage by approximately half compared to 16-bit, providing substantial efficiency gains while preserving model accuracy.
- **4-bit (INT4)** significantly shrinks the model size and memory requirements, allowing for deployment on fewer resources; however, it can introduce noticeable accuracy degradation unless carefully managed with advanced quantization methods.

1 Laboone, Maxime. "**Introduction to Weight Quantization**." *towards data science*, 7 July 2023.

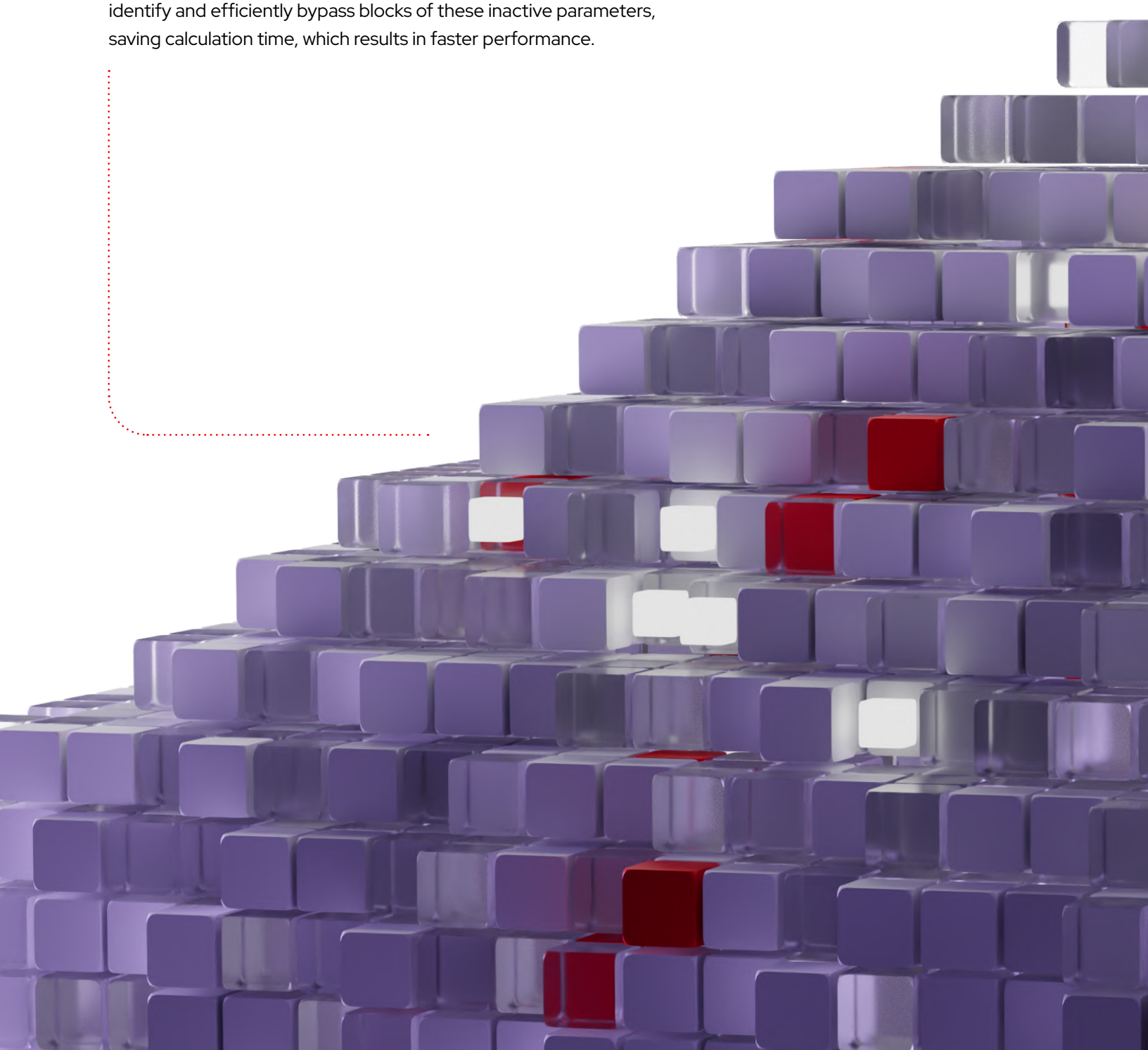
2 "**AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration**." GitHub, accessed 8 Aug. 2025.

3 Turganbay, Raushan. "**Unlocking Longer Generation with Key-Value Cache Quantization**." *Hugging Face*, 16 May 2024.

Reducing computational load with sparsity

Sparsity reduces computational demands by intentionally setting some of the model's parameters to 0, allowing systems to bypass unnecessary operations—like skipping blank sections on a form. This improves speed and efficiency without needing to fully retrain the model.

2:4 sparsity is **a structured approach** that sets exactly 2 out of every 4 parameters to 0, allowing specialized hardware to quickly identify and efficiently bypass blocks of these inactive parameters, saving calculation time, which results in faster performance.

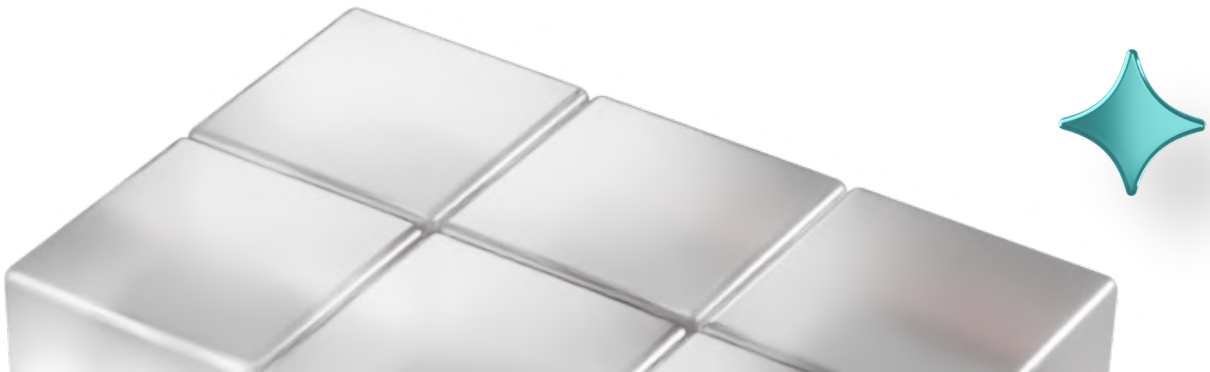


The evolution of **large language models**

Large language models (LLMs), primarily built upon transformer architectures that use self-attention mechanisms to model relationships across sequences of text, have evolved from research experiments to foundational tools powering real-world applications. Their scale—often reaching 10s or 100s of billions of parameters—allow for high levels of reasoning, creativity, and domain specificity. They do so through a process called inference.

Inference is the procedure by which a trained model processes new input data and generates an output, such as predicting the next word in a sentence or identifying an object in an image. Unlike training, which involves learning from large datasets, inference is focused on applying that learned knowledge to make real-time decisions. As such, inference must be fast and efficient, especially when models are deployed in production environments to support interactive applications, real-time analysis, or large-scale automation.

Inference models process input data such as text, images, or audio as tokens, passing them through multilayer transformer architectures to generate predictions. Tokens are the discrete units into which input data is broken before being processed by a model. In text-based models, tokens can represent individual characters, subwords, or entire words, depending on the tokenization strategy used.

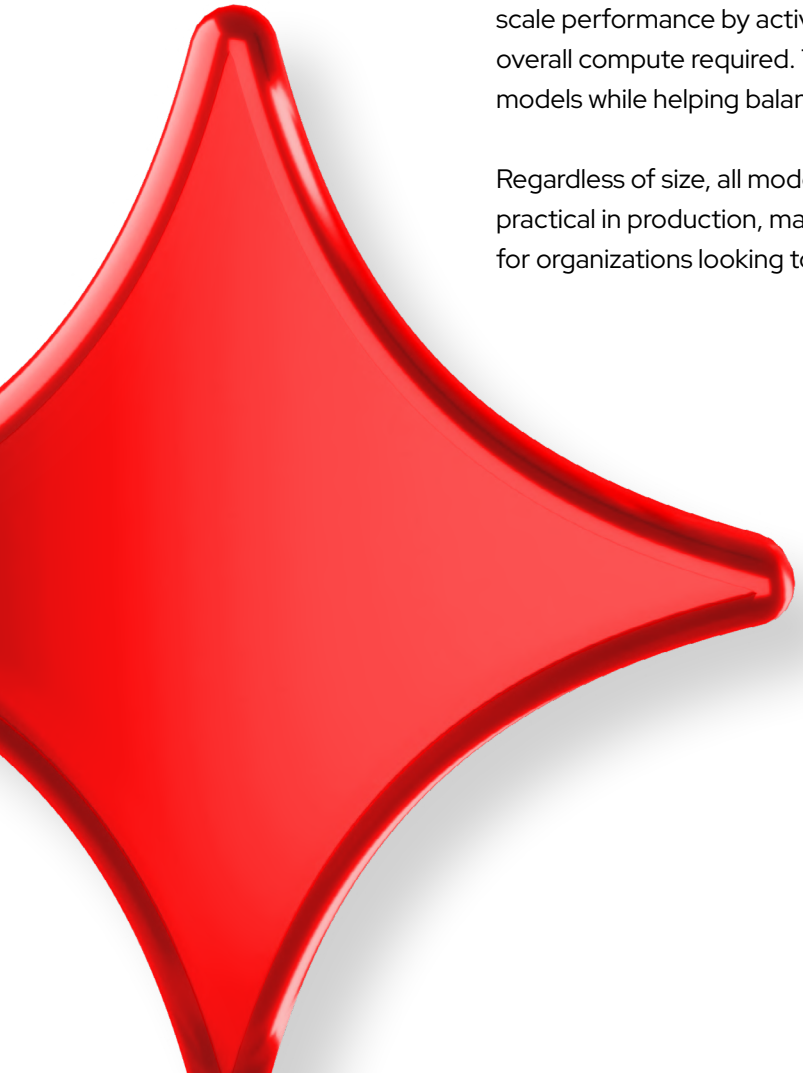


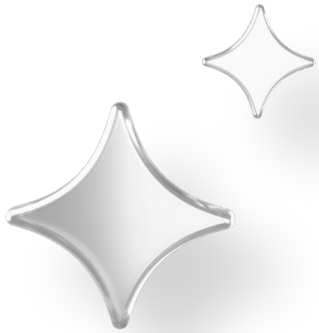
These models pass input tokens through deep, multilayer transformer architectures that apply a sequence of mathematical operations to analyze context, weigh relationships, and determine likely outputs. Each layer refines the model's understanding of the input, ultimately producing a prediction, 1 token at a time. This step-by-step token generation allows for highly accurate and contextually appropriate outputs, but also contributes to the computational intensity of inference workloads, especially for large models with many layers.

Beyond text-based LLMs, similar architectures now underpin a range of AI domains, including vision models and multimodal systems. Vision models apply the same principles of token-based transformer computation to images and video. Instead of breaking text into tokens, pixel data is converted into embeddings. These embeddings capture spatial patterns, edges, textures, and relationships between visual elements, allowing the model to perform tasks such as image classification, object detection, segmentation, and visual question answering. When deployed in production, vision models can support use cases such as automated inspection, medical imaging, and content moderation.

As organizations adopt AI more broadly, model architectures continue to grow in size and complexity. New approaches such as mixture of experts (MoE) aim to scale performance by activating only parts of the model per inference, reducing the overall compute required. These innovations open the door to even more powerful models while helping balance performance with cost and energy demands.

Regardless of size, all models require efficient serving and optimization to be practical in production, making inference performance engineering a critical priority for organizations looking to deploy models.





Challenges of **inference serving**

Serving inference for large models presents several challenges.

Models with billions of parameters require substantial GPU memory to store their weights and intermediate states, such as key-value (KV) caches. As the number of concurrent requests or the length of the inputs increase, memory constraints become critical bottlenecks, limiting the model's throughput and responsiveness. Basic serving methods often struggle with inefficient batching techniques, leading to underutilized hardware resources and increased latency.

Furthermore, implementations of attention mechanisms in transformer architectures can be computationally intensive, especially with long inputs, which significantly slows down response times. Addressing these challenges demands sophisticated runtime optimizations such as efficient memory management, advanced batching strategies, and optimized attention mechanisms such as paged attention. Together, these help to increase performance and responsiveness in real-world applications.

A full-stack approach to **inference performance**

Inference optimization refers to the process of improving how efficiently an AI model runs once deployed in production. Running LLMs in production can quickly become expensive, especially when dealing with high token volumes, long prompts, and growing usage demands. **Cost optimization in inference comes down to reducing memory consumption, increasing throughput, efficiently balancing and routing inference traffic, and minimizing hardware needs, all without sacrificing accuracy or user experience.**

While model training is generally a single-instance task (barring exceptions for model retraining), inference happens constantly, generating real-time outputs in response to user inputs. For LLMs and vision models, inference can quickly become the most expensive and resource-intensive part of an AI deployment, especially when scaled across hybrid or global infrastructure.

Effectively serving LLMs at scale requires a comprehensive, full-stack optimization strategy that addresses both the model itself and the serving runtime. While we are primarily looking at optimizing model parameters through **quantization** and **sparsity**, additional performance gains can be realized by refining the inference serving process through techniques such as **chunked prefill**,[4] **prefix caching**,[5] **speculative decoding**,[6] and **disaggregated prefill and decode**.[7]

4 "**Optimization and Tuning**." *vLLM*, 7 Aug. 2025.

5 "**What is Automatic Prefix Caching?**" *vLLM*, accessed 8 Aug. 2025.

6 "**How Speculative Decoding Boosts vLLM Performance by up to 2.8x**." *vLLM*, 17 Oct. 2024.

7 Du, Kuntai. "**vLLM Office Hours - Disaggregated Prefill and KV Cache Storage in vLLM - November 14, 2024**." *YouTube*, 18 Nov. 2024.

Overview of inference runtimes and model formats

Since basic runtimes are a bottleneck, serving large models efficiently requires choosing the right inference runtime. Popular runtimes include:

- **vLLM**. Virtual large language model is a library of open source code maintained by the vLLM community. It helps LLMs perform calculations more efficiently and at scale. Specifically, vLLM is an inference server that speeds up the output of gen AI applications by making better use of the GPU memory. It is widely adopted across the industry because of its superior throughput and low-latency performance, helped by innovations like paged attention, which allows more tokens to be processed efficiently in GPU memory.

- **Triton**. Often mistaken for a standalone runtime, Triton functions more as a front-end application programming interface (API) for various backend engines, including TensorRT and vLLM. While Triton with TensorRT may offer slightly higher performance on NVIDIA GPUs, it comes at the cost of increased set-up complexity and limited model support. Customers often report that achieving performance gains with Triton takes significantly more effort than with vLLM.

- **SGLang**. A newer entry, SGLang is derived from vLLM, and optimized for specific use cases. It uses many of the same underlying components as vLLM but supports fewer model architectures. While it may outperform vLLM in narrow contexts, its limited flexibility and community support make it less attractive for broad enterprise adoption.

A dual approach to model efficiency

1: Optimizing the inference runtime (vLLM)

Runtime limitations

As mentioned in the last chapter, **serving LLMs efficiently can be challenging due to limitations inherent in basic inference serving methods**.

These runtime limitations include inefficient GPU memory usage, suboptimal batch processing, and slow token generation. Runtimes typically store intermediate computation data, such as KV caches, inefficiently, consuming extensive GPU memory and limiting the capacity for concurrent requests. Furthermore, simplistic batching strategies can leave GPUs idle or underutilized, significantly reducing throughput. Additionally, basic runtimes struggle with slow attention mechanisms, causing extended latency when handling long input sequences.

Why vLLM

vLLM addresses many runtime challenges by providing advanced techniques specifically optimized for inference performance:

- **Continuous batching**. vLLM minimizes GPU idle time by concurrently processing tokens from multiple incoming requests. Instead of handling a single request at a time, it groups tokens from different sequences into batches, significantly improving GPU utilization and inference throughput.

- **PagedAttention**. vLLM uses a novel memory management strategy called PagedAttention, efficiently handling large-scale KV caches. This technique dynamically allocates and manages GPU memory pages, greatly increasing the number of concurrent requests and supporting significantly longer sequences without memory bottlenecks.

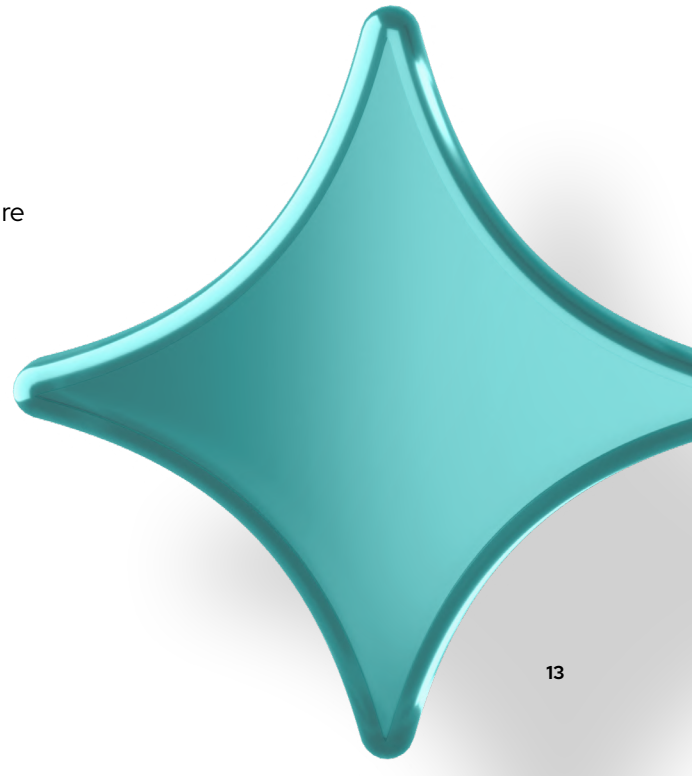
For an in-depth exploration, read this technical blog about vLLM.

vLLM deployment benefits

Comprehensive integration capabilities: vLLM can directly load models from popular repositories such as Hugging Face and serves as a high-performance backend within frameworks like Triton Inference Server. Its compatibility with a wide variety of hardware platforms, including NVIDIA GPUs, AMD GPUs, and Google TPUs, further simplifies enterprise-scale deployment.

Standardization and vendor agnosticism: By using a widely adopted runtime such as vLLM, organizations gain standardization benefits, which support reliable performance across diverse hardware environments and avoid lock-in to proprietary solutions.

For a deeper understanding of vLLM's parallelism techniques, visit this technical deep dive blog.



2: Optimizing the AI model

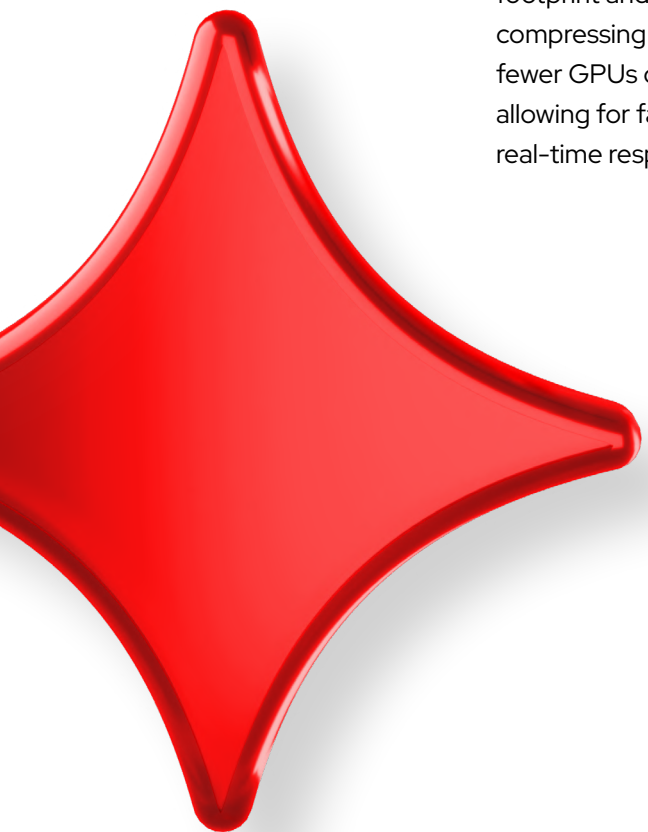
The importance of optimizing large language models

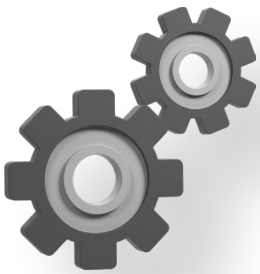
One of the main challenges in production is managing memory and compute efficiency. Large models often require vast amounts of GPU memory to store parameters and context in the KV cache, particularly when dealing with long prompts or multiple concurrent requests. If models aren't optimized, they can run inefficiently, leading to higher operational costs. Latency is another critical concern: users expect real-time responses, and delays caused by large model size or inefficient execution can negatively affect experience and the efficacy of downstream workflows.

Why compress a model

Compressing a model helps address some of the most significant challenges organizations face when deploying AI at scale: cost efficiency and performance optimization.

As models increase in size to billions of parameters, serving them in production becomes resource-intensive, demanding extensive memory and compute power. Model compression techniques, including quantization and sparsity, slightly reduce the precision and number of parameters, while significantly lowering the memory footprint and compute requirements without substantially sacrificing accuracy. By compressing models, organizations can run AI workloads more efficiently, using fewer GPUs or other accelerators, thus dramatically cutting operational costs and allowing for faster inference, which is essential for applications that require real-time responses.





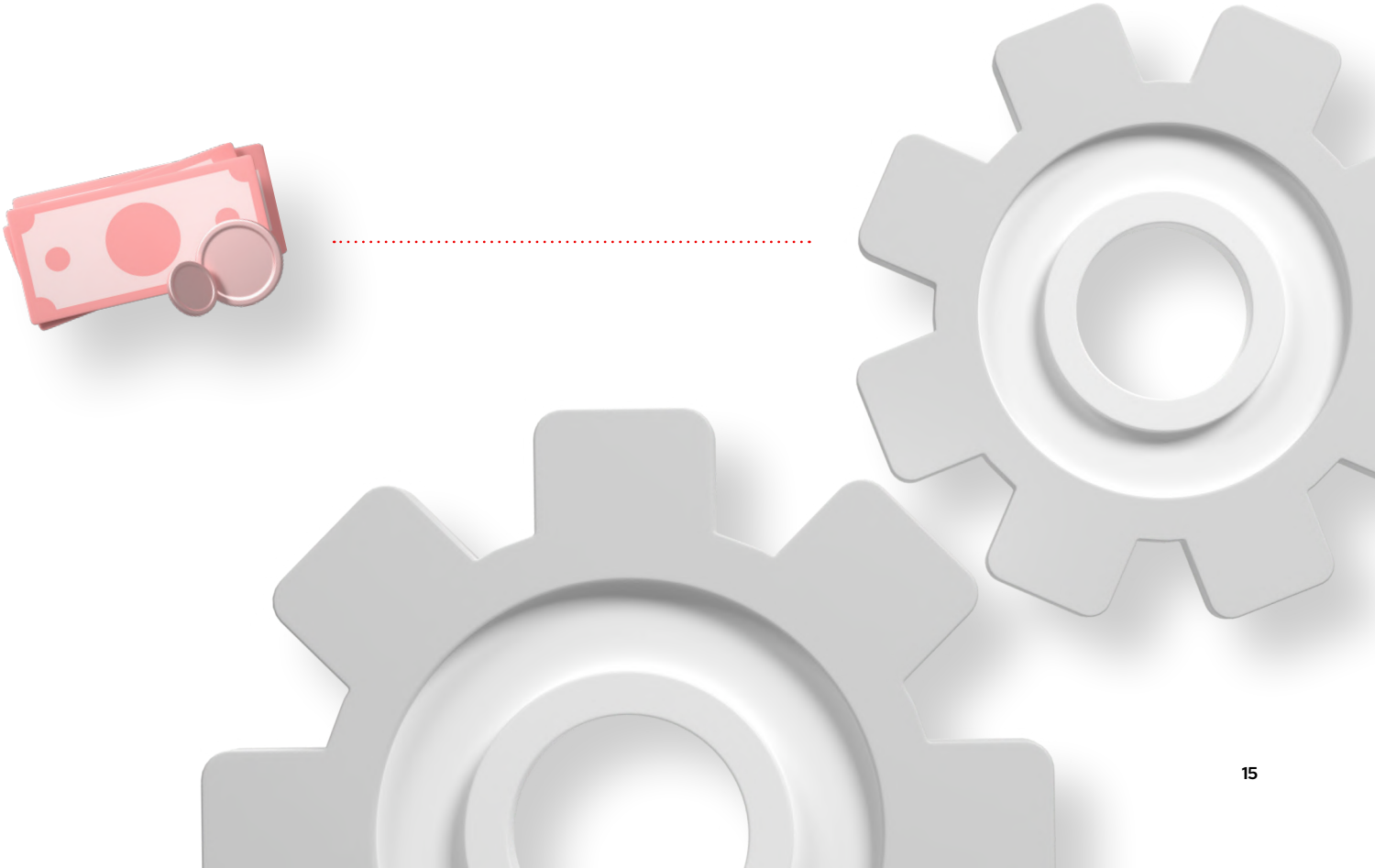
How can my model be cost-optimized for inference?

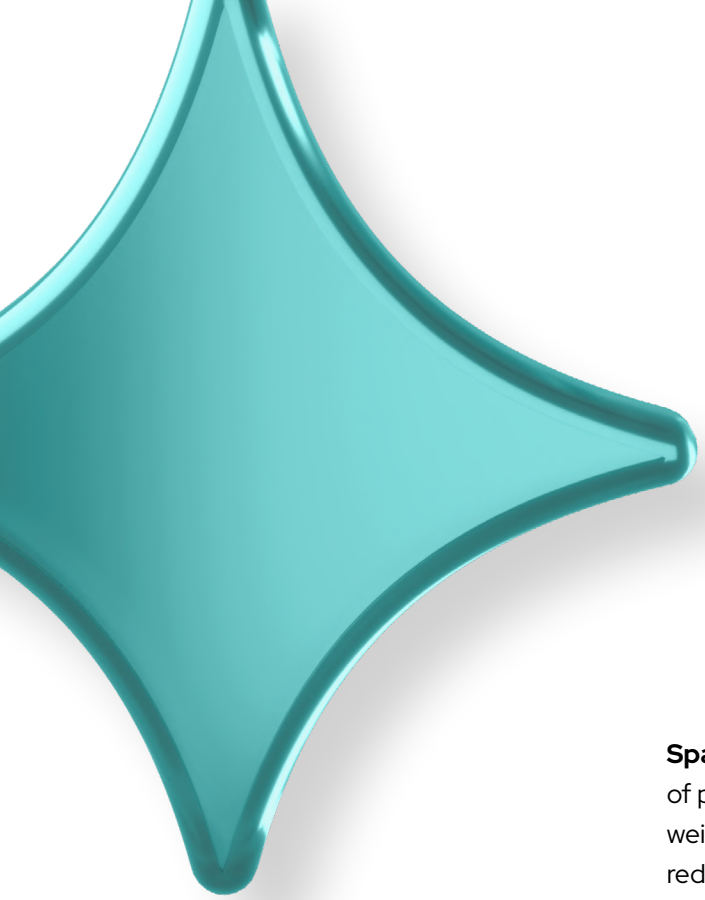
One of the most effective ways to reduce these costs is to compress your model. Compression techniques such as quantization and sparsity shrink model size and reduce compute requirements, allowing inference workloads to run on fewer or smaller GPUs.

Quantization optimizes a model by reducing the precision of its numerical values—specifically, the model's weights and activations. Typically, models operate at 16-bit precision (or even 32-bit precision), using formats such as FP16 or BF16.

Quantization compresses these values to lower precision formats such as 8-bit (INT8 or FP8) or even 4-bit integers (INT4). This process significantly reduces the memory needed to store model parameters, allowing models such as a 70-billion parameter Llama to shrink from approximately 140 GB to as low as 40 GB. Such reductions not only free up memory for additional computations, but also enhance throughput, especially in memory-bound situations. For example, a GPU with 48 GB of VRAM will handle a 40 GB model faster than a 140 GB one.

However, aggressive quantization can affect accuracy due to precision loss. To mitigate this, fine-grained quantization employs scaling factors that preserve model accuracy, often achieving less than 1% degradation. Quantization can double computational throughput by optimizing hardware usage, thus significantly decreasing latency and operational costs.





Sparsity optimizes a model by introducing a structured reduction of parameters—essentially setting a large proportion of the model's weights to 0. This technique works by identifying and eliminating redundant or less critical weights, simplifying computations during inference. Sparsity can substantially reduce model complexity, thereby decreasing memory usage and computational load, allowing faster inference and lower operational costs.

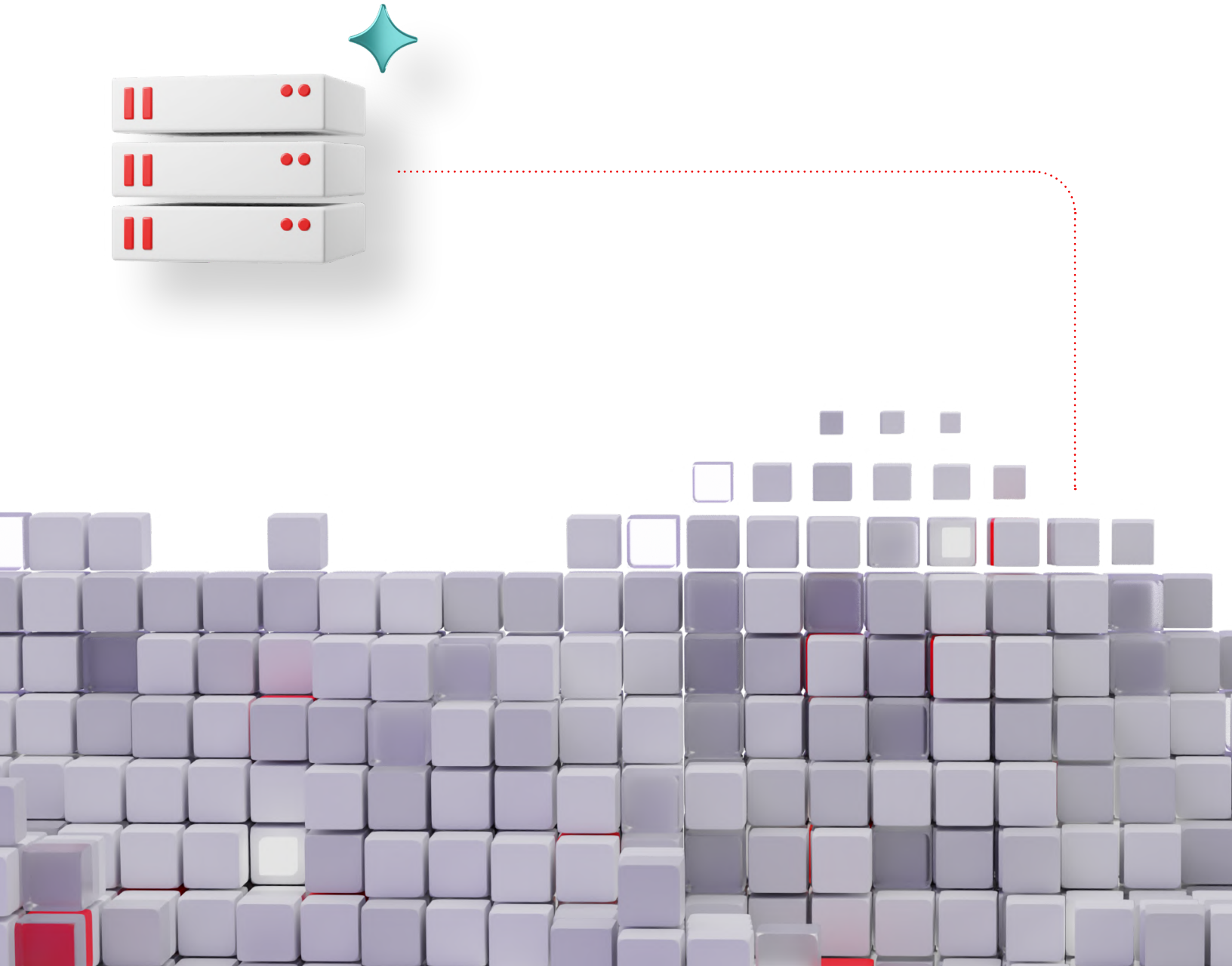
However, achieving sparsity effectively requires retraining the model—a computationally intensive step that demands significant upfront resources. Sparsity's efficiency depends on the hardware capabilities, such as semistructured sparsity supported by modern accelerators like GPUs, where specific patterns of zeroed weights enable faster computations. The key advantage is its ability to reduce computational requirements significantly when properly implemented.

While sparsity can yield notable benefits, particularly when combined with other optimization methods like quantization, it typically requires a more involved optimization process. Therefore, it's recommended for scenarios with extensive scale or specialized hardware setups. By carefully applying sparsity, organizations can improve inference efficiency, but due to the complexity involved, quantization is more commonly recommended as the primary optimization technique.

By adopting compression workflows and validated runtimes, organizations can better manage operational costs, support scalability, and prepare for future increases in AI usage without overcommitting infrastructure resources.

Will accuracy be compromised?

While model compression techniques like quantization and sparsity reduce memory and compute requirements, they are specifically designed to maintain acceptable levels of accuracy. For example, 8-bit quantization typically delivers near-baseline accuracy while halving memory consumption. Even 4-bit models can retain strong performance when optimized using advanced quantization techniques like weight rounding and calibration. Structured sparsity patterns, such as 2:4 sparsity, allow hardware accelerators to skip redundant operations without degrading output quality. In many production scenarios, teams achieve significant resource savings with minimal or no reduction in model performance. Testing and validation remain essential, but for most applications, well-implemented compression yields high-efficiency inference with accuracy intact.

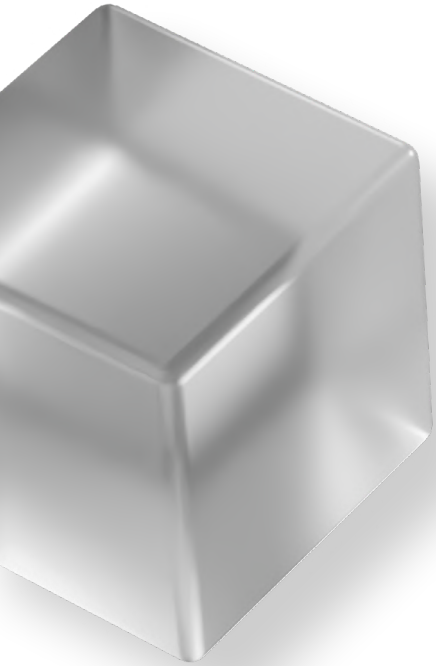


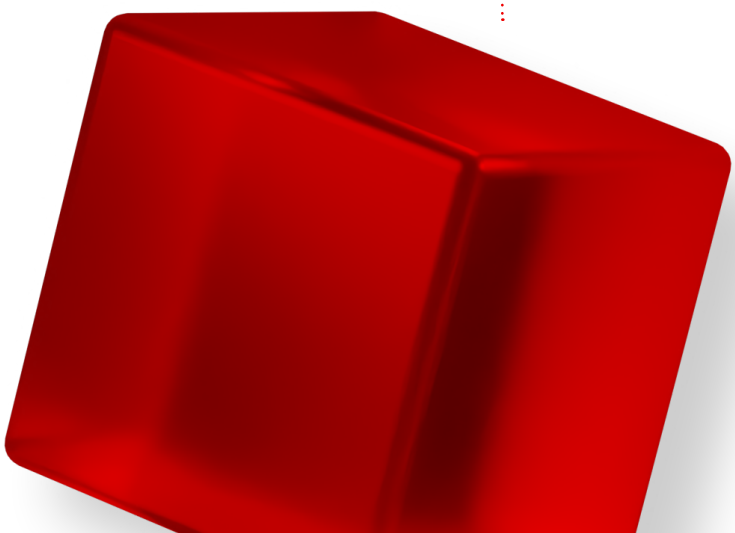
Red Hat AI

What is Red Hat AI?

Red Hat AI is a platform that accelerates AI innovation and reduces the operational cost of developing and delivering AI solutions across hybrid cloud environments. It simplifies integration with private data, helps reduce costs with optimized models and efficient inference, and accelerates delivery of agentic AI workflows with a scalable, flexible platform.

Red Hat AI allows organizations to manage and monitor the lifecycle of both predictive and gen AI models at scale, from single-server deployments to highly scaled distributed platforms. The platform is powered by open source technologies and a partner ecosystem focused on performance, stability, and GPU support across various infrastructures.





Red Hat AI includes includes:

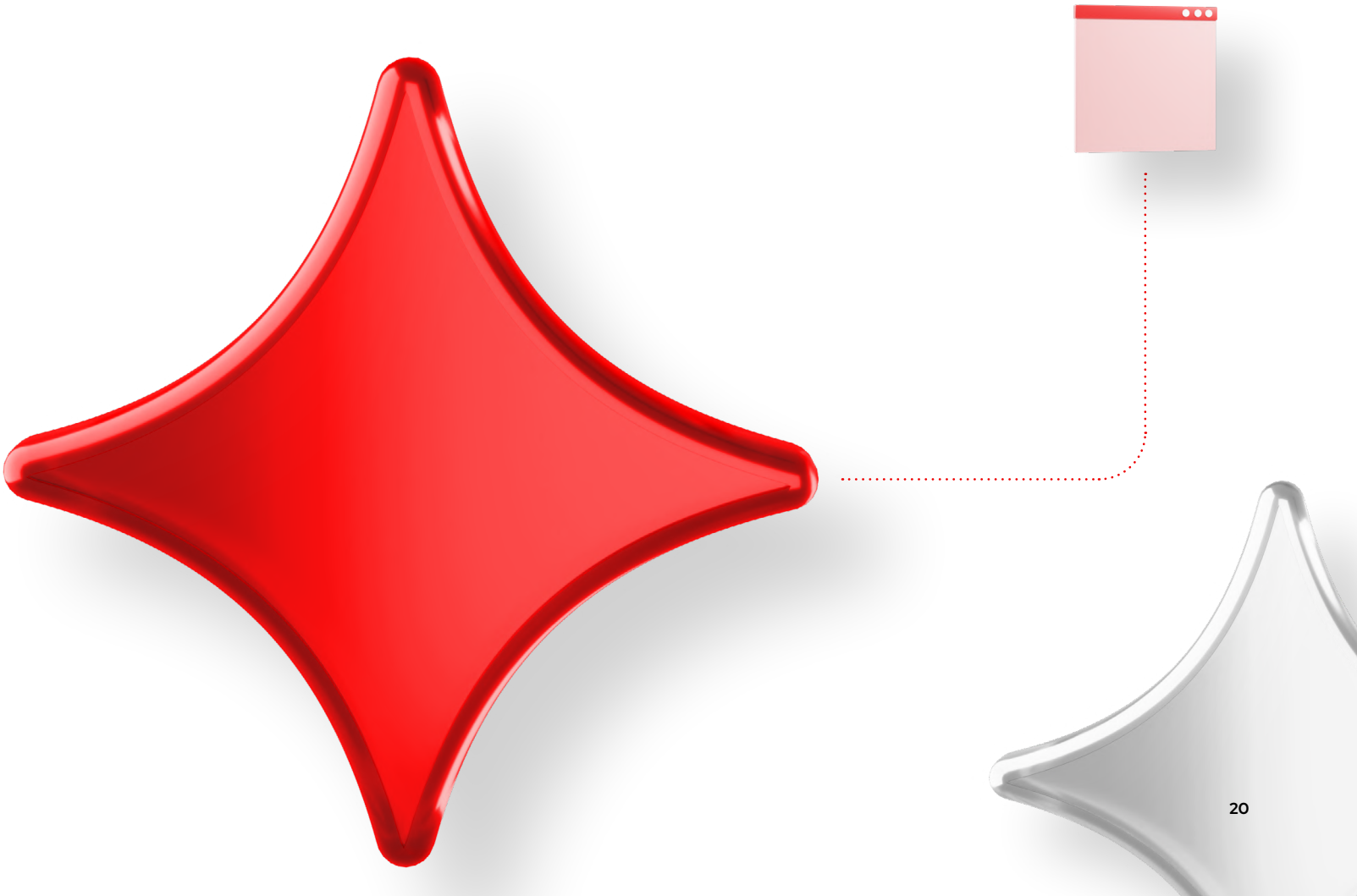
- **Optimized and validated models.** Pre-evaluated and performance-tested models to reduce the burden of testing and fine-tuning.
- **LLM Compressor.** A toolkit that helps users apply quantization and compression to popular models, reducing resource requirements for inference without losing accuracy.
- **Model customization.** Tools to fine-tune or adapt foundation models to specific enterprise needs.
- **High-performance inference runtime.** An optimized vLLM-based runtime using advanced batching and memory management techniques for efficient, scalable, and reliable model serving.
- **Distributed inference.** Powered by llm-d, it provides the fleet-wide orchestration needed to efficiently balance and distribute inference requests beyond single-server deployments.
- **LLMOps.** Practices and tools that streamline the deployment, monitoring, and management of LLMs in production environments.
- **AI safety and evaluations.** Frameworks and methodologies for assessing model accuracy, fairness, and robustness, making sure AI deployments are responsible and reliable.
- **Flexible and consistent scaling.** Infrastructure support that makes sure there is flexibility and consistency when scaling AI across hybrid cloud environments.
- **Accelerated agentic AI delivery.** Capabilities designed to rapidly deploy advanced, autonomous AI systems, keeping organizations at the forefront of AI innovation.

Optimizing inference with Red Hat AI

Red Hat AI helps organizations optimize inference through advanced techniques designed to balance efficiency, accuracy, and cost-effectiveness.

Red Hat AI emphasizes 2 primary aspects of inference optimization: a high-performant integrated stack, powered by vLLM and llm-d, and optimized models. By combining these approaches, Red Hat's AI portfolio delivers rapid inference performance while reducing the required computational resources. Specifically, **Red Hat AI Inference** employs continuous batching memory-efficient methods, KV-cache routing and disaggregated prefill and decode to process more tokens per second, achieving greater throughput with less GPU usage.

Red Hat AI LLM Compressor provides a standardized approach to applying the compression techniques discussed in this e-book, and aims to deliver optimization with 99% retained accuracy. It helps users generate optimized versions of popular models that are tuned for inference runtimes such as vLLM. This makes it easier to run high-performing, compressed models on a wider variety of hardware setups.



Red Hat AI provides **extensive validation** to help organizations confidently select, deploy, and scale optimized models. Given the wide range of available LLMs, organizations often face challenges identifying models that best align with their use cases in terms of accuracy, performance, and cost efficiency. To address these challenges, Red Hat AI uses open-source validation tools (such as GuideLLM, Language Model Evaluation Harness, and vLLM) to rigorously benchmark model performance across multiple evaluation tasks. This validation supports reproducibility and informed model selection, reducing complexity and uncertainty.

Red Hat AI also offers **capacity guidance** to help organizations accurately plan AI infrastructure and optimize the use of resources, addressing common issues such as hardware underuse, high compute costs, and inefficiencies at inference time. This combination of validated models, optimized deployment settings, and tailored hardware recommendations allows organizations to enhance flexibility, accelerate deployments, and achieve predictable performance while managing costs effectively.

With compression techniques and optimized runtimes, Red Hat AI makes it practical to deploy LLMs at scale, equipping teams to meet rising demands while maintaining control over cost, complexity, and the use of compute assets.

Next steps

Ready to reduce the cost and complexity of serving LLMs? Learn more about Red Hat AI Inference or connect with your Red Hat representative to get started.