Red Hat Summit

Connect

# Eigenes LLM hosten

## mit dem Red Hat vLLM Inference Server

Adrian Luberda
Account Solution Architect

Ingo Boernig
Chief Architect

Marta Blaszczyk
Associate Account Solution Architect

Adrian Luberda

Account Solution Architect

Ingo Boernig

Chief Architect

Marta Blaszczyk

Associate Account Solution Architect

# Agenda

Introduction to the topic:

60-min hands-on lab

# Challenges

### Memory

*LLMs can consume huge amounts of GPU memory and inefficient allocation often forces teams to overprovision costly hardware to keep models running.*

### Latency

*Generating each word is compute-heavy, so when more people use the model at once, it has to share the same GPUs and responses get slower.*

### Scaling

*To serve many users, LLMs often need multiple GPUs working together, which makes the system more complex and expensive to build and run.*

# Build the fastest, easiest-to-use open-source LLM inference & serving engine

# Solutions

## Continuous Batching

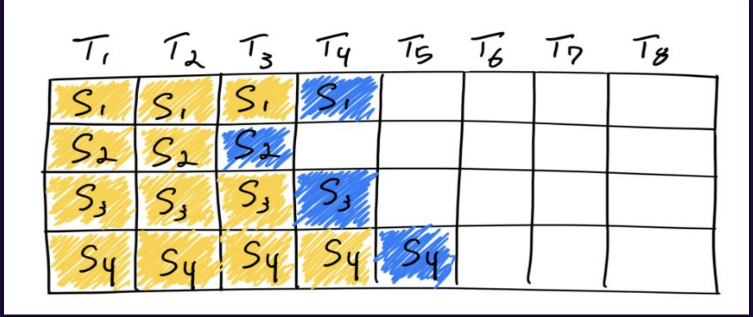
*vLLM lets new requests join and finished requests leave an active batch on every step, keeping the GPU constantly busy and giving faster responses even when many users are hitting the model at once.*

## Paged Attention

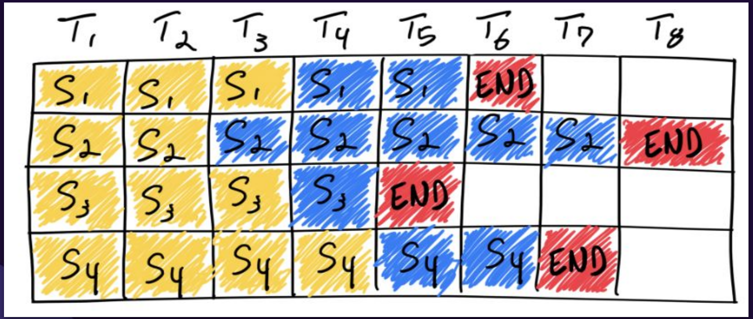
*LLM splits the model's "memory" into small reusable blocks so it only keeps what it really needs on the GPU, dramatically reducing wasted memory and letting each GPU handle more and longer requests.*

# Challenge 1: Batching

## Static Batching 🙅🙅🙅

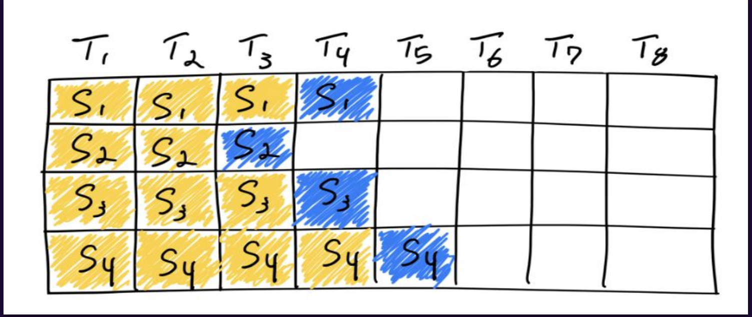


| $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ |
|---|---|---|---|---|---|---|---|
| $S_1$ | $S_1$ | $S_1$ | $S_1$ | | | | |
| $S_2$ | $S_2$ | $S_2$ | | | | | |
| $S_3$ | $S_3$ | $S_3$ | $S_3$ | | | | |
| $S_4$ | $S_4$ | $S_4$ | $S_4$ | $S_4$ | | | |

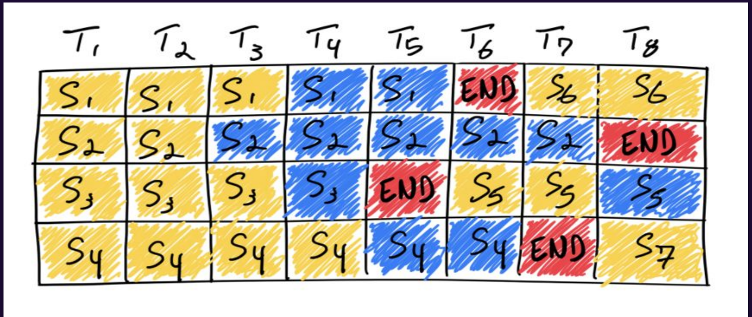


| $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ |
|---|---|---|---|---|---|---|---|
| $S_1$ | $S_1$ | $S_1$ | $S_1$ | $S_1$ | END | | |
| $S_2$ | $S_2$ | $S_2$ | $S_2$ | $S_2$ | $S_2$ | $S_2$ | END |
| $S_3$ | $S_3$ | $S_3$ | $S_3$ | END | | | |
| $S_4$ | $S_4$ | $S_4$ | $S_4$ | $S_4$ | $S_4$ | END | |

## Continuous Batching 🙏🙏🙏



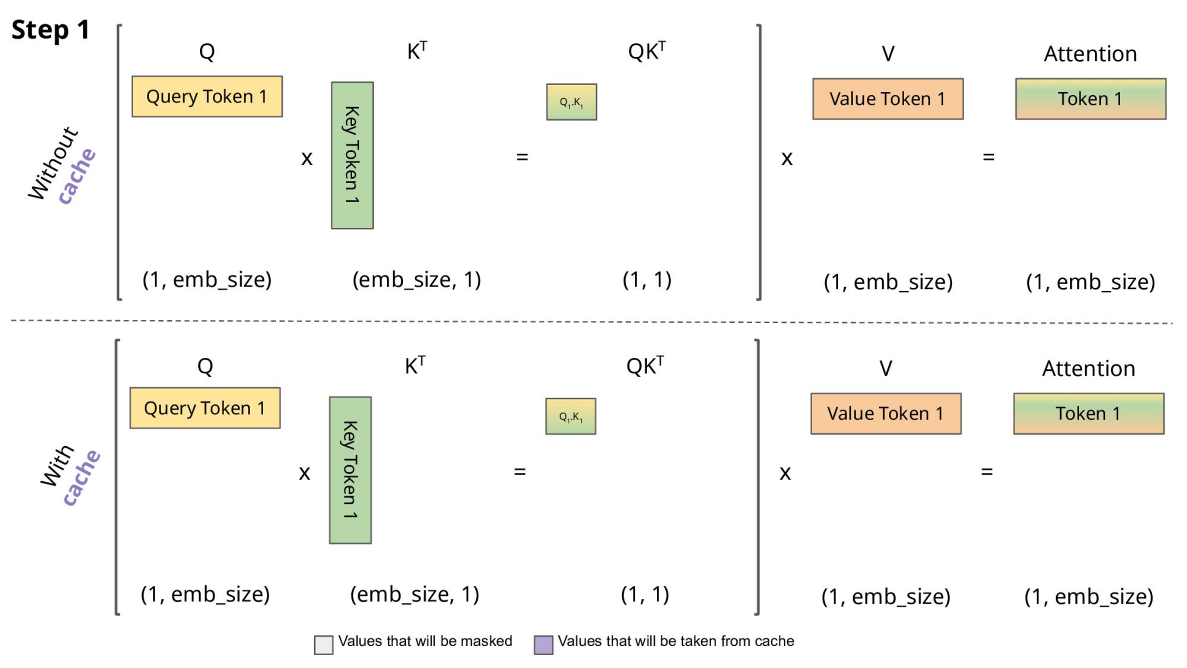
| $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ |
|---|---|---|---|---|---|---|---|
| $S_1$ | $S_1$ | $S_1$ | $S_1$ | | | | |
| $S_2$ | $S_2$ | $S_2$ | | | | | |
| $S_3$ | $S_3$ | $S_3$ | $S_3$ | | | | |
| $S_4$ | $S_4$ | $S_4$ | $S_4$ | $S_4$ | | | |



| $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ |
|---|---|---|---|---|---|---|---|
| $S_1$ | $S_1$ | $S_1$ | $S_1$ | $S_1$ | END | $S_6$ | $S_6$ |
| $S_2$ | $S_2$ | $S_2$ | $S_2$ | $S_2$ | $S_2$ | $S_2$ | END |
| $S_3$ | $S_3$ | $S_3$ | $S_3$ | END | $S_5$ | $S_5$ | $S_5$ |
| $S_4$ | $S_4$ | $S_4$ | $S_4$ | $S_4$ | $S_4$ | END | $S_7$ |

# **Challenge 2**: KV Caching

**KV Cache**: Caching Key and Value vectors in self-attention saves redundant computation and accelerates decoding



**Step 1**

Without cache

Q: Query Token 1 (1, emb_size) x $K^T$: Key Token 1 (emb_size, 1) = $QK^T$: $Q_1.K_1$ (1, 1)

x V: Value Token 1 (1, emb_size) = Attention: Token 1 (1, emb_size)

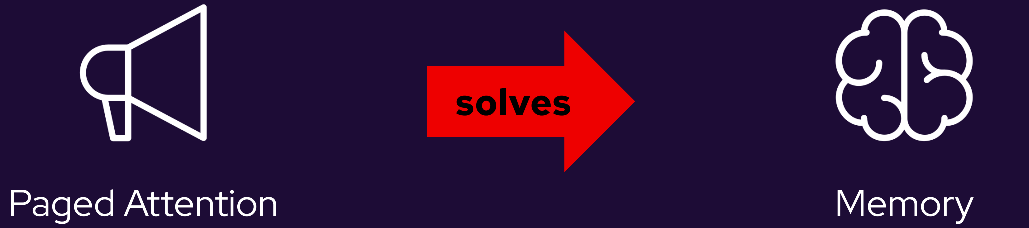
With cache

Q: Query Token 1 (1, emb_size) x $K^T$: Key Token 1 (emb_size, 1) = $QK^T$: $Q_1.K_1$ (1, 1)

x V: Value Token 1 (1, emb_size) = Attention: Token 1 (1, emb_size)

□ Values that will be masked ▨ Values that will be taken from cache

# Solution 1: PagedAttention



Paged Attention

solves

Memory

# Solution 2: Continuous Batching

Continuous Batching

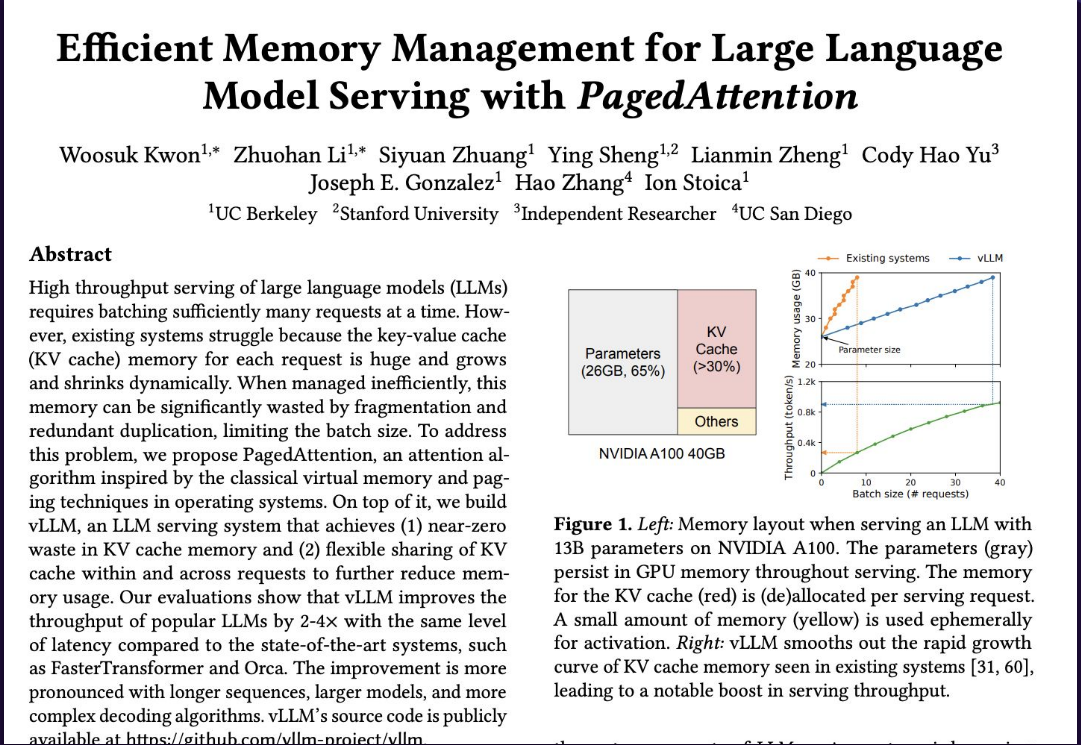
solves

Latency

+

Scaling

Paper



# Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Woosuk Kwon[1,*] Zhuohan Li[1,*] Siyuan Zhuang[1] Ying Sheng[1,2] Lianmin Zheng[1] Cody Hao Yu[3]
Joseph E. Gonzalez[1] Hao Zhang[4] Ion Stoica[1]
[1]UC Berkeley [2]Stanford University [3]Independent Researcher [4]UC San Diego

## Abstract

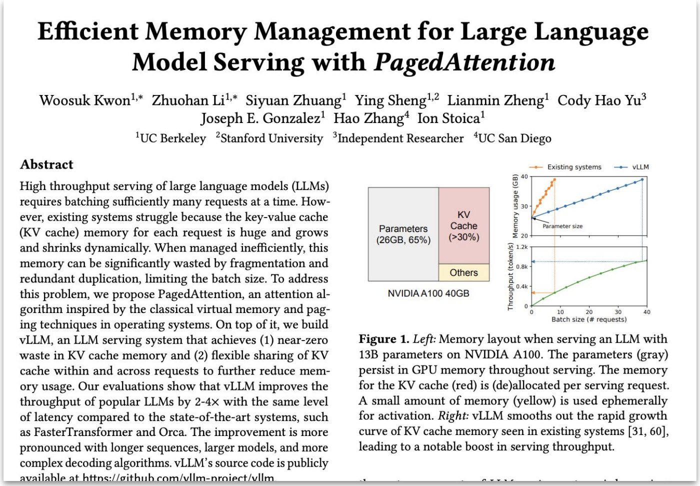
High throughput serving of large language models (LLMs) requires batching sufficiently many requests at a time. However, existing systems struggle because the key-value cache (KV cache) memory for each request is huge and grows and shrinks dynamically. When managed inefficiently, this memory can be significantly wasted by fragmentation and redundant duplication, limiting the batch size. To address this problem, we propose PagedAttention, an attention algorithm inspired by the classical virtual memory and paging techniques in operating systems. On top of it, we build vLLM, an LLM serving system that achieves (1) near-zero waste in KV cache memory and (2) flexible sharing of KV cache within and across requests to further reduce memory usage. Our evaluations show that vLLM improves the throughput of popular LLMs by 2-4× with the same level of latency compared to the state-of-the-art systems, such as FasterTransformer and Orca. The improvement is more pronounced with longer sequences, larger models, and more complex decoding algorithms. vLLM's source code is publicly available at https://github.com/vllm-project/vllm.

**Figure 1.** *Left:* Memory layout when serving an LLM with 13B parameters on NVIDIA A100. The parameters (gray) persist in GPU memory throughout serving. The memory for the KV cache (red) is (de)allocated per serving request. A small amount of memory (yellow) is used ephemerally for activation. *Right:* vLLM smooths out the rapid growth curve of KV cache memory seen in existing systems [31, 60], leading to a notable boost in serving throughput.

https://arxiv.org/abs/2309.06180

# 2 Year Journey Of vLLM

vLLM has rapidly evolved from a research project to the open source default



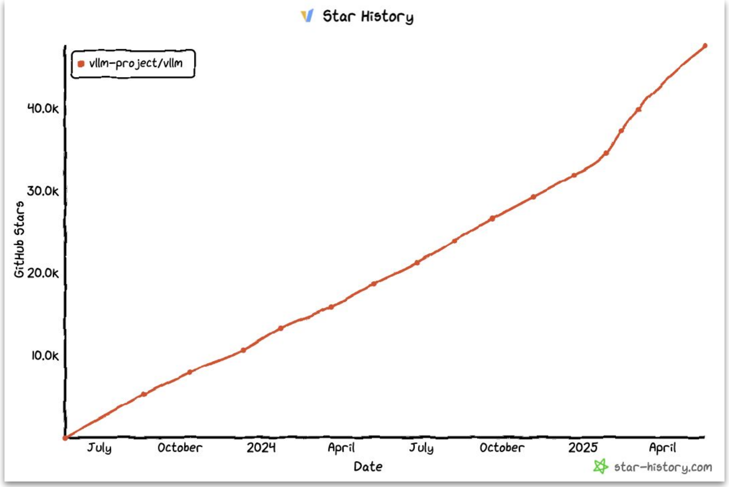
## Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Woosuk Kwon[1,*] Zhuohan Li[1,*] Siyuan Zhuang[1] Ying Sheng[1,2] Lianmin Zheng[1] Cody Hao Yu[3] Joseph E. Gonzalez[1] Hao Zhang[4] Ion Stoica[1]
[1]UC Berkeley [2]Stanford University [3]Independent Researcher [4]UC San Diego

### Abstract

High throughput serving of large language models (LLMs) requires batching sufficiently many requests at a time. However, existing systems struggle because the key-value cache (KV cache) memory for each request is huge and grows and shrinks dynamically. When managed inefficiently, this memory can be significantly wasted by fragmentation and redundant duplication, limiting the batch size. To address this problem, we propose PagedAttention, an attention algorithm inspired by the classical virtual memory and paging techniques in operating systems. On top of it, we build vLLM, an LLM serving system that achieves (1) near-zero waste in KV cache memory and (2) flexible sharing of KV cache within and across requests to further reduce memory usage. Our evaluations show that vLLM improves the throughput of popular LLMs by 2-4× with the same level of latency compared to the state-of-the-art systems, such as FasterTransformer and Orca. The improvement is more pronounced with longer sequences, larger models, and more complex decoding algorithms. vLLM's source code is publicly available at https://github.com/vllm-project/vllm.

**Figure 1.** *Left:* Memory layout when serving an LLM with 13B parameters on NVIDIA A100. The parameters (gray) persist in GPU memory throughout serving. The memory for the KV cache (red) is (de)allocated per serving request. A small amount of memory (yellow) is used ephemerally for activation. *Right:* vLLM smooths out the rapid growth curve of KV cache memory seen in existing systems [31, 60], leading to a notable boost in serving throughput.



- **Pervasive** → 100k daily installs in 2025; 50k GitHub stars
- **Explosive Growth** → 10x usage increase in 2024
- **Vibrant Community** → 1000+ contributors

# Who Uses vLLM?

vLLM is the de-facto inference OSS inference server with ~600k  weekly installs and ~50k GitHub stars

- **Model as a Service:** AWS, GCP, Azure, NVIDIA, ...
- **AI in Scaled Production:** Amazon, Microsoft, LinkedIn, Meta, ...
- **Proprietary Deployments:** Snowflake, Roblox, IBM, ...
- **Foundation Model Labs:** Meta, Mistral, Qwen, Cohere, ...
- **Fine-tuning Frameworks:** veRL, TRL, OpenRLHF, ...
- **Hardware Platforms:** NVIDIA, AMD, Google, Intel, ARM, ...

Red Hat
Enterprise Linux AI

# It provides a package version of the vLLM inference server project that allows you to serve LLMs across hybrid cloud.

# Red Hat AI Inference Server

Gain consistent, fast and cost-effective inference at scale

## Inference runtime for the hybrid cloud

Run your models of choice across any accelerator and any environment

## Compress Models

Reduce compute and costs while preserving accuracy

## Red Hat AI Hugging Face repository

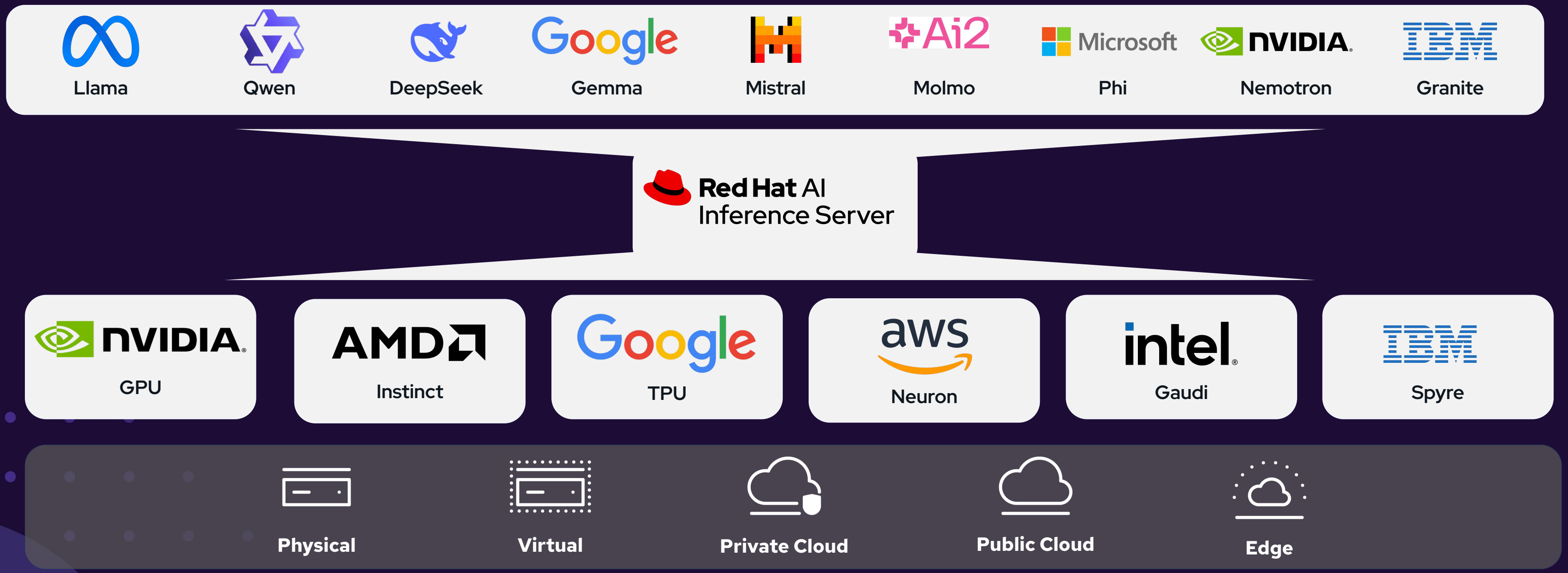
Access a collection of third-party validated and optimized models ready for inference.

## Certified for all Red Hat products

Deployable across non-Red Hat Linux and Kubernetes platforms

Red Hat AI Inference Server is included in OpenShift AI and RHEL AI

## Single platform to run any model, on any accelerator, on any cloud



Llama · Qwen · DeepSeek · Gemma · Mistral · Molmo · Phi · Nemotron · Granite

Red Hat AI Inference Server

NVIDIA GPU · AMD Instinct · Google TPU · aws Neuron · intel Gaudi · IBM Spyre

Physical · Virtual · Private Cloud · Public Cloud · Edge

Agenda

# What you are going to see

1

Deploy the LLMs on your infrastructure

2

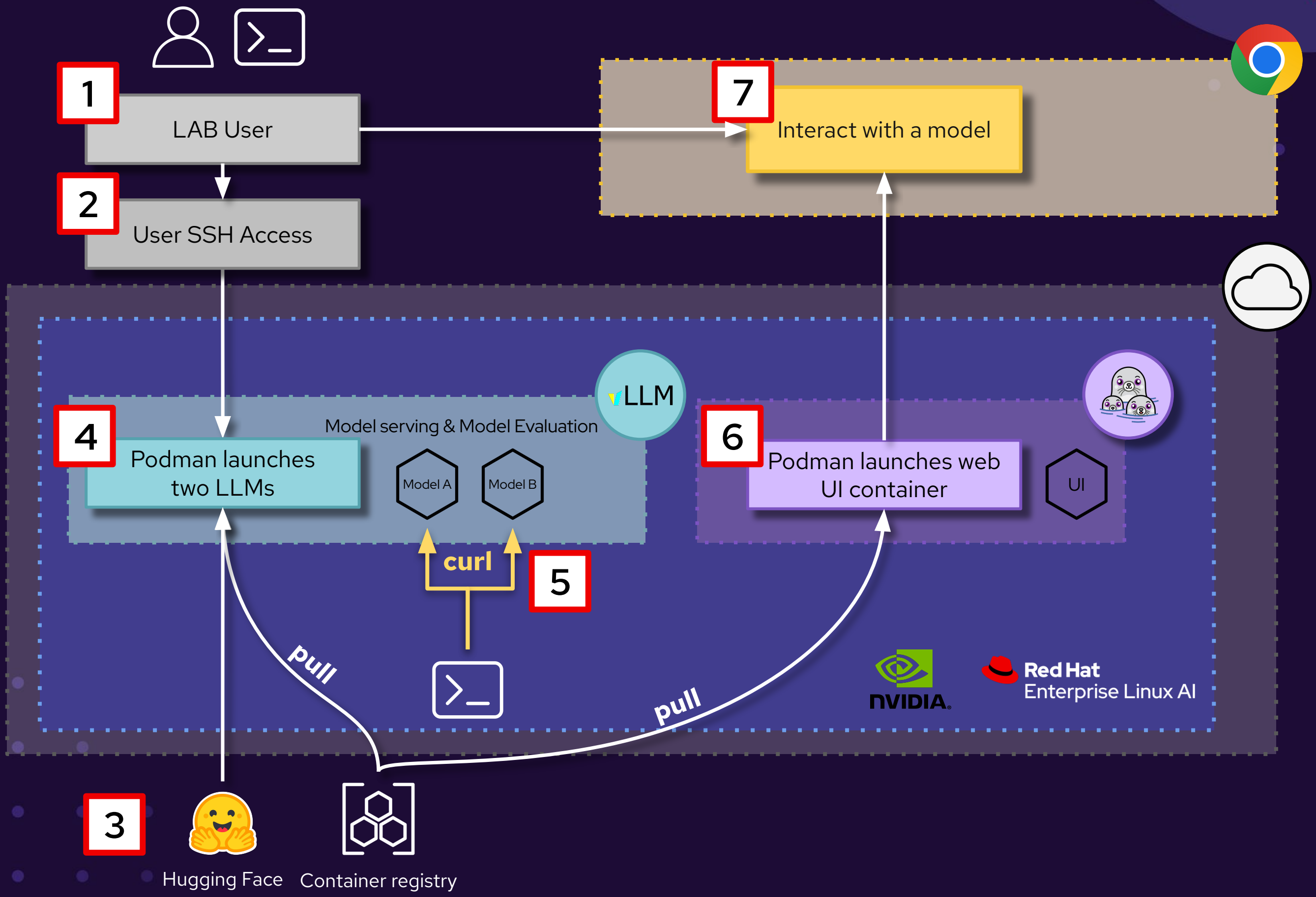
Interact with the models through CLI
(Optionally via web UI)

3

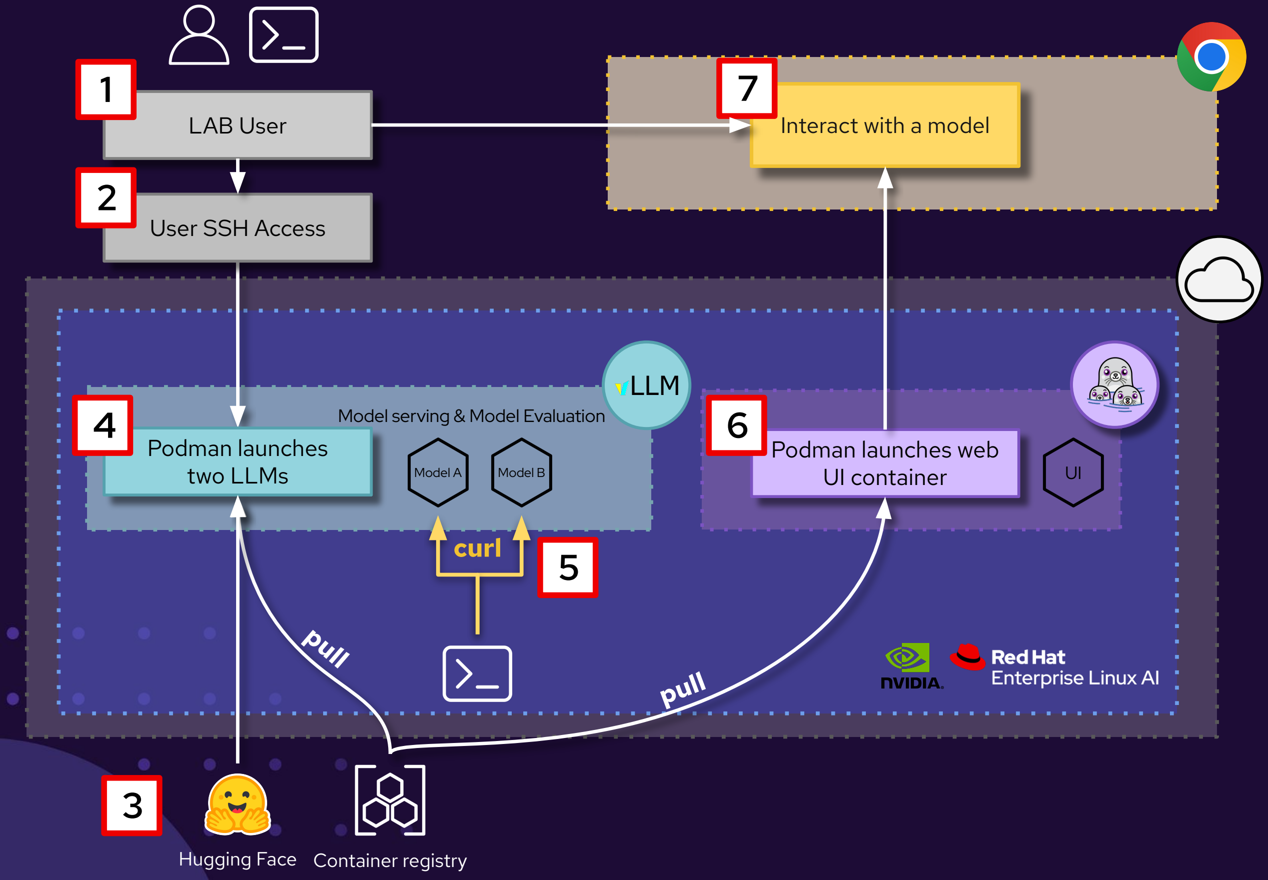
A vs B

Compare two different optimized variants of one model

Diagram


1
LAB User
2
User SSH Access
7
Interact with a model
vLLM
Model serving & Model Evaluation
4
Podman launches two LLMs
Model A
Model B
curl
5
6
Podman launches web UI container
UI
pull
pull
NVIDIA
Red Hat
Enterprise Linux AI
3
Hugging Face
Container registry

## Diagram



1. Accessing the RHEL AI environment via SSH and verifying GPU, Podman and registries
2. Log in to the Red Hat Registry and pull the Red Hat AI Inference Server container image (vllm-cuda-rhel9:3.0.0)
3. Start the server running Model A to enable model download and inference
4. Call the model using the OpenAI-style HTTP API (curl to port 8000).
5. Run a second, differently optimized variant of the model (quantized W8A8)
6. Use the llm-eval-test tool
7. (Optional) Deploy a simple Web UI container to interact with the model via a browser.

Instructions

# red.ht/vllm

## How do we do it?

### Find a group

*Form a team of 2–3 people and sit together at one table.*

### Log in with your e-mail

*Go to the workshop URL*

### Follow the documentation

*Use the link we provide and work through the lab steps together as a team.*

# Workshop instructions:

red.ht/vllm

# llm-eval-harness

- The LM Evaluation Harness is a unified framework designed for the evaluation of generative language models (LLMs).
- It was originally **developed by EleutherAI**.
- It is part of OpenShift AI
- One of the featured that is being assessed is accuracy

https://www.eleuther.ai/projects/large-language-model-evaluation

# Jetzt Session bewerten!

Einfach QR-Code scannen,
Session aus der Liste wählen
und bewerten. **Vielen Dank!**

**red.ht/rhsc-darmstadt-feedback**