KubeCon | CloudNativeCon

North America 2024

KubeCon CloudNativeCon North America 2024

Load-Aware GPU Fractioning for LLM Inference on Kubernetes

Olivier Tardieu & Yue Zhu, IBM Research

Bird’s-Eye View

- What?
  - Reduce the cost of running AI workloads on Kubernetes
- How?
  - Pack multiple Large Language Model (LLM) servers on the same GPU

Enable GPU fractioning in Kubernetes

Right-size workloads

https://sched.co/1izuH

Today’s talk

Outline

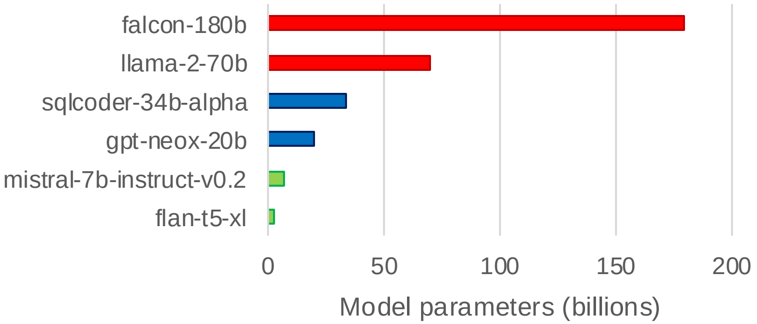
- Motivation
  - Right-sizing Large Language Models (LLMs)
  - Kubernetes and Multi-Instance GPUs (MIG)
- Modeling LLMs
  - Insights from Performance Profiling
  - Performance Model
  - Model Results
- AutoFit
  - Design and Implementation
  - Demo

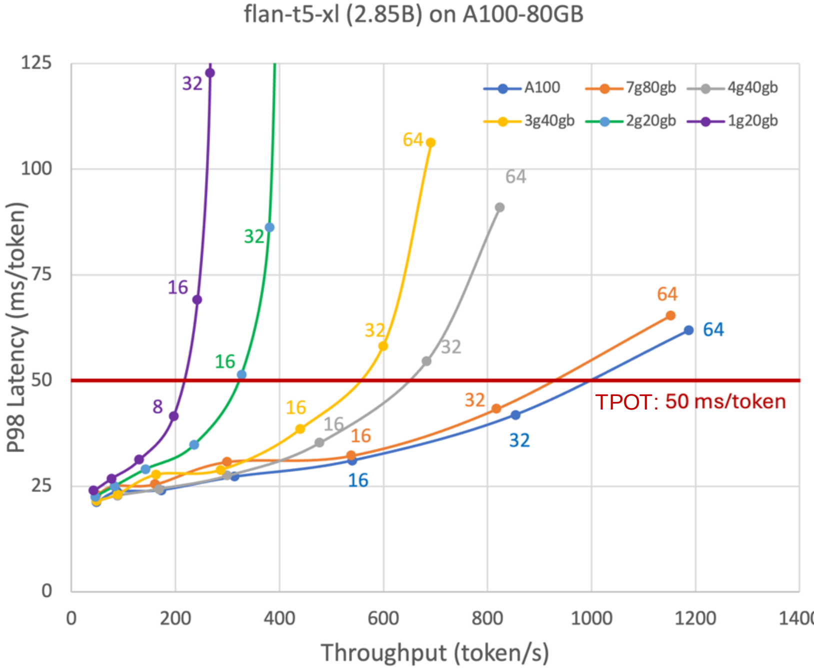
Large Language Models

- Inference server
  - Compute/memory requirements depend on model ***and load***

⇒ Packing opportunity

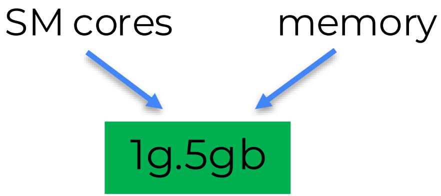
⇒ **Right-sizing challenge**

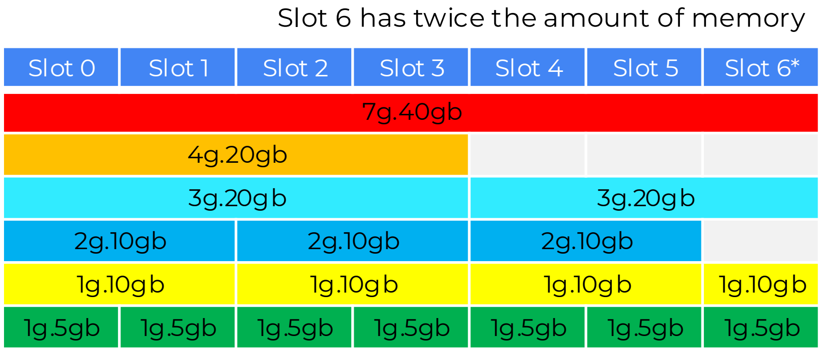




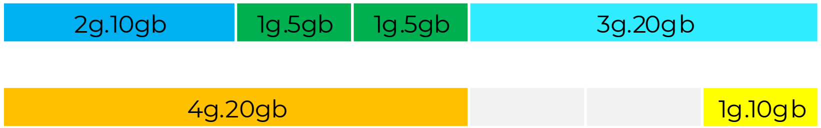
Multi-Instance GPU

- Partition GPU
  - Application sees a smaller GPU
  - Full isolation, no code change
  - Up to 7 slices
  - Small number of profiles
  - Profiles can be mixed
  - Incremental slice creation and deletion





NVIDIA A100-40GB MIG profiles



Example A100-40GB MIG layouts

MIG in Kubernetes Today

- NVIDIA GPU operator
  - Mature API
  - Admin preselects a layout
  - Nodes offer MIG slices

```
allocatable:
  nvidia.com/mig-1g.5gb: 2
  nvidia.com/mig-2g.10gb: 1
  nvidia.com/mig-3g.20gb: 1
```

  - Pods request MIG slices

```
requests:
  limits:
    nvidia.com/mig-1g.5gb: 1
```

- InstaSlice (IBM/Red Hat)
  - Experimental API
  - No need to preselect a layout
  - Slices are created on demand
  - Pod request MIG slices

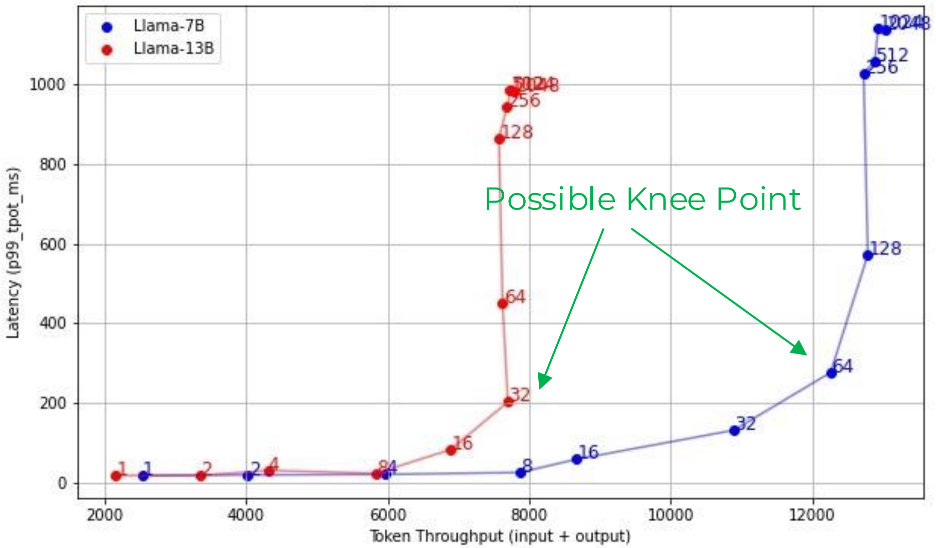
```
requests:
  limits:
    nvidia.com/mig-1g.5gb: 1
```

Modeling LLMs

What do we learn from the profiling study?

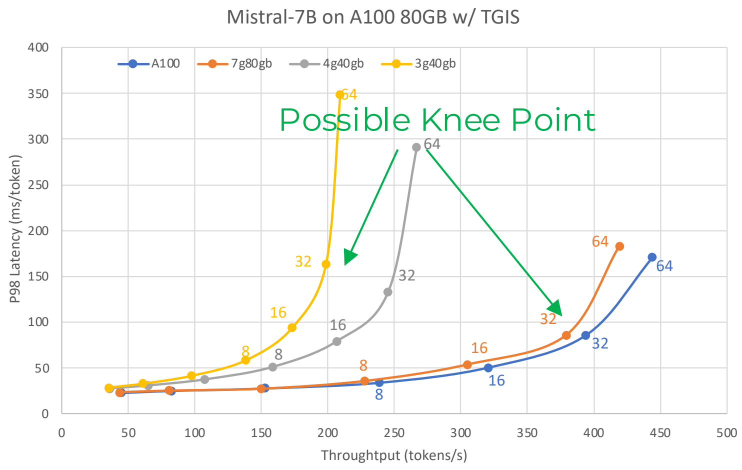
- With fixed input length, throughput keeps constant, but latency sharply increases, when increasing concurrent requests
  - There exists a knee point on the performance curve

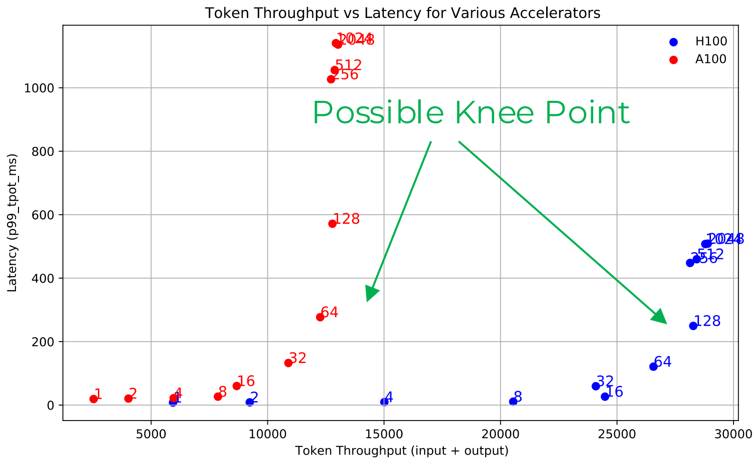
Throughput vs Latency on A100 via vLLM



What do we learn from the profiling study?

- Similarly, for a given GPU/MIG type, we also see throughput and latency hit a “performance wall” when increasing request load
  - This is caused by the intersection of GPU computation and memory bandwidth limit



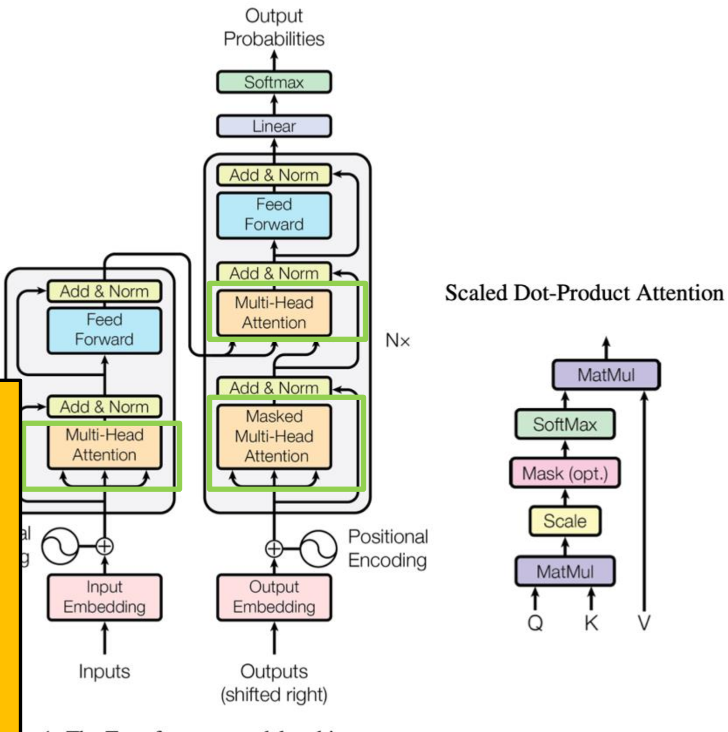


Attention Layers in LLM

- **Attention Layers Dominate**: Transformers are built with multiple stacked attention layers
  - Attention layers allow the model to focus the input, weighing relevance across all words
- **Query (Q)**, **Key (K)**, **Value (V)** in the Attention Layer

We use Attention Layers to estimate throughput via **model's Arithmetic Intensity**

- $Intensity_{model} = \frac{FLOPS/s}{Bytes\ Moved/s}$
- $Intensity_{model} = \frac{Cost_{Comp}\ of\ Attention\ Layers}{Cost_{Mem} of\ Attention\ Layers}$
- $Throughput = Intensity_{model} \times BW_{mem}$



re 1: The Transformer - model architecture.

[1] Vaswani, A. "Attention is all you need." *Advances in Neural Information Processing Systems* (2017).

How do we estimate throughput?

- The ***AI*** of LLM model at decoding (for FP16)

Line in Algorithm (Byte)	Load from Memory	Compute (FLOPS)	Store to Memory (Byte)
Line 1	input + Weights: FP16 size of $(X + W^Q + W^K + W^V)$	cost_matmul: $3 \times (X \times W^Q)$	FP16 size of (Q, K, V)
	$2 \times (B \times SL \times N + N \times N \times 3)$	$2 \times 3 \times (B \times SL \times N \times N)$	$2 \times 3 \times (B \times SL \times N)$
Line 2	FP16 size of Q and K	cost_matmul: $Q \times K^T$	FP16 size of S
	$2 \times (B \times SL \times N + B \times (SL + cache\_len) \times N)$	$2 \times (B \times SL \times N \times (SL + cache\_len))$	$2 \times (B \times SL \times (SL + cache\_len))$

$$Cost_{Comp} = B \times \overbrace{\boxed{(6N^2 + 2 \times (1 + cache\_len) \times (3 + 2N))}}^{a}$$

$$Cost_{Mem} = B \times \underbrace{\boxed{2(7N + (1 + cache\_len) \times (4 + N))}}_{b} + \underbrace{\boxed{6N^2}}_{c}$$

For a given sequence length (L),

$$Throughput_L(B) = \frac{Cost_{comp}}{Cost_{Mem}} \times BW_{mem}$$

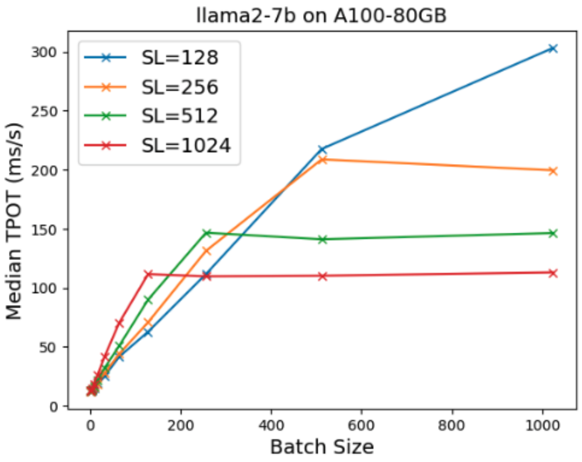
No peak BW

The **knee point** of the model corresponds to the knee point of this function when it is fitted to the given profiling data points.

$$\boxed{Throughput_L(B) = \frac{a \times B}{b \times B + c}}$$

How do we estimate latency (TPOT)?

- Example: llama2-7b on A100-80GB
  - Linear increase in latency with rising batch size, before reaching the “knee point”
  - Large sequence length is also easier to reach the "knee point”



- Before the “knee point”, for a given sequence length, latency is linear to the batch size (concurrent request count)

$$Latency_L = num\_layer \times \left(\frac{Cost_{mem}}{BW_{mem}} + \frac{Cost_{comp}}{BW_{mem}} \right)$$

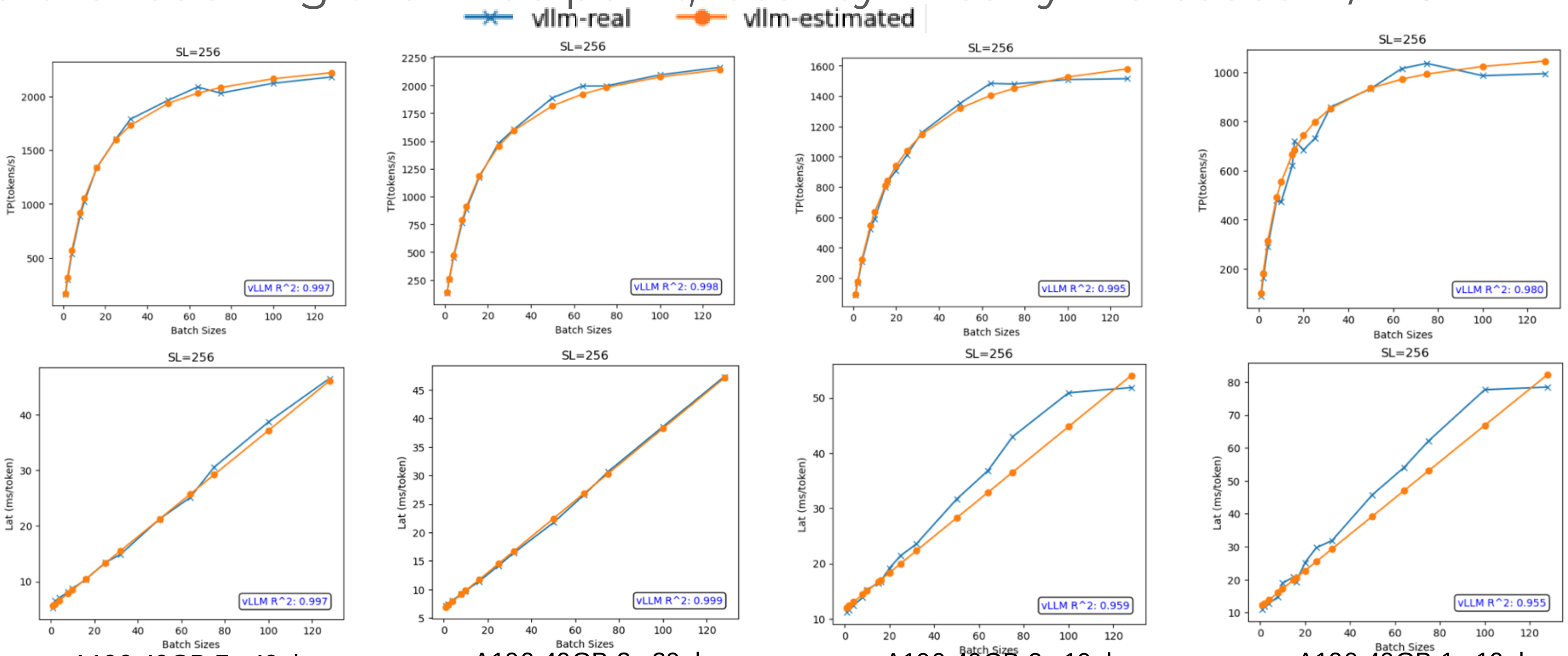
$$Latency_L = num\_layer \times (a \times B + c)$$

Perf Model Results: OPT-1.3B

- Measured TP match predicted TP
- Before reaching the knee point, latency linearly increases w/ BS

Throughput

Latency

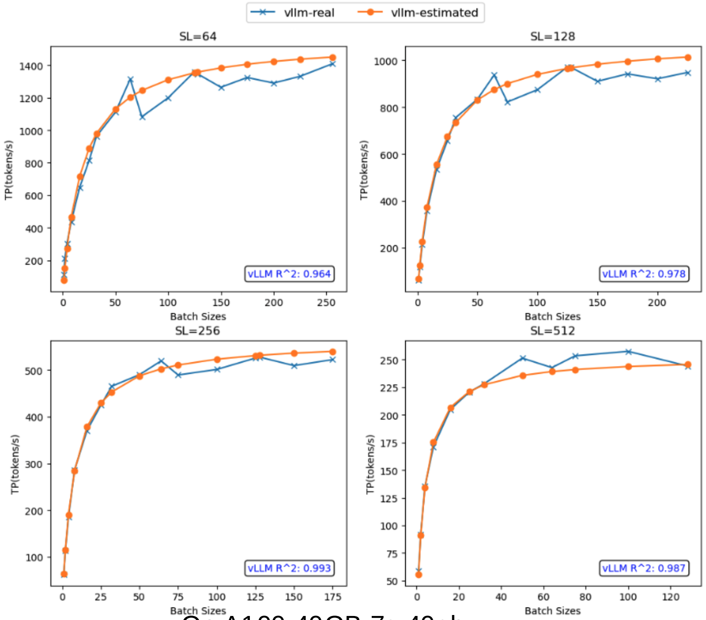


A100-40GB 7g.40gb | A100-40GB 3g.20gb | A100-40GB 2g.10gb | A100-40GB 1g.10gb

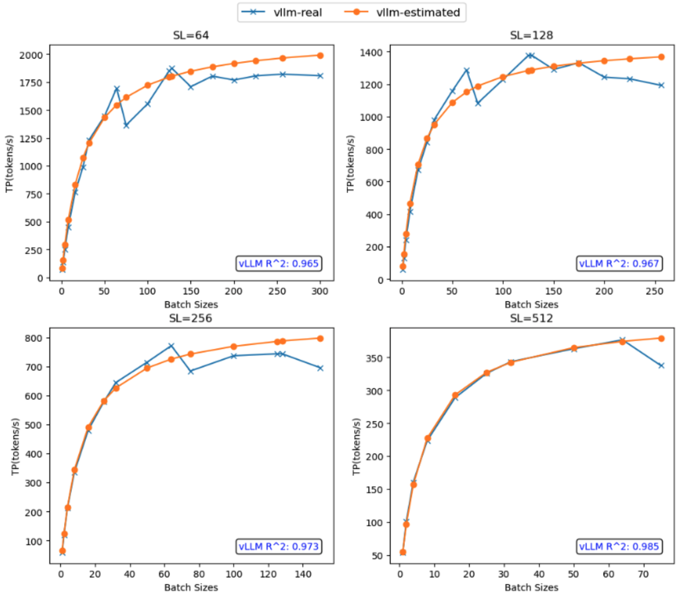
[1] All experiments measured via vLLM-0.6.3

Perf Model Results: Mistral-7B-Q Throughput

- Mistral-7B-Instruct-v0.3-quantized.w8a16
  - Despite some variations in TP, overall trend remains consistent w/ initial estimation



On A100-40GB 7g.40gb

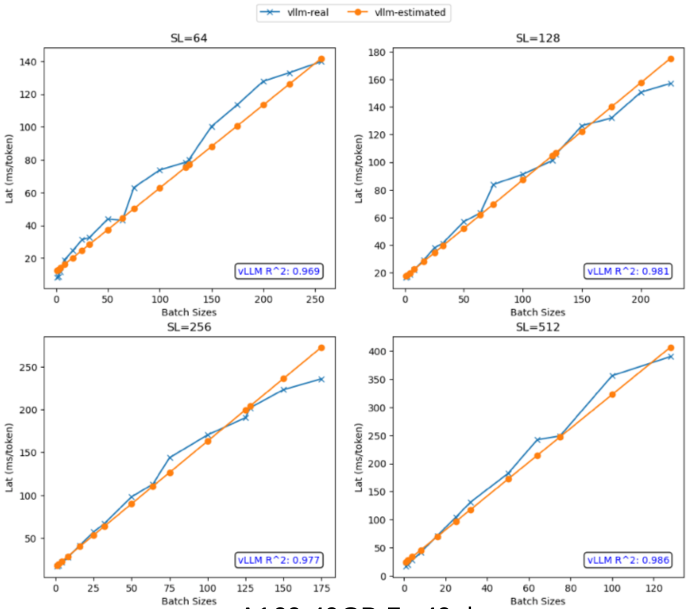


on A100-40GB 3g.20gb

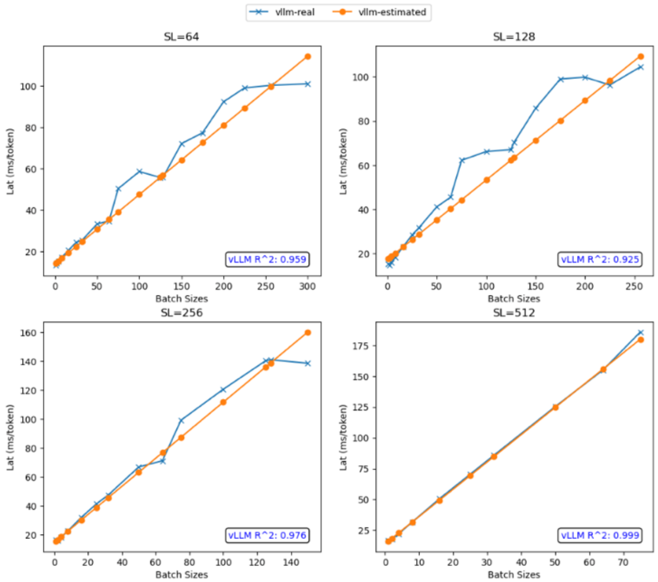
[1] All experiments measured via vLLM-0.6.3

Perf Model Results: Mistral-7B-Q Latency

- Mistral-7B-Instruct-v0.3-quantized.w8a16
  - Despite some variations in TP, overall trend remains consistent w/ initial estimation



on A100-40GB 7g.40gb

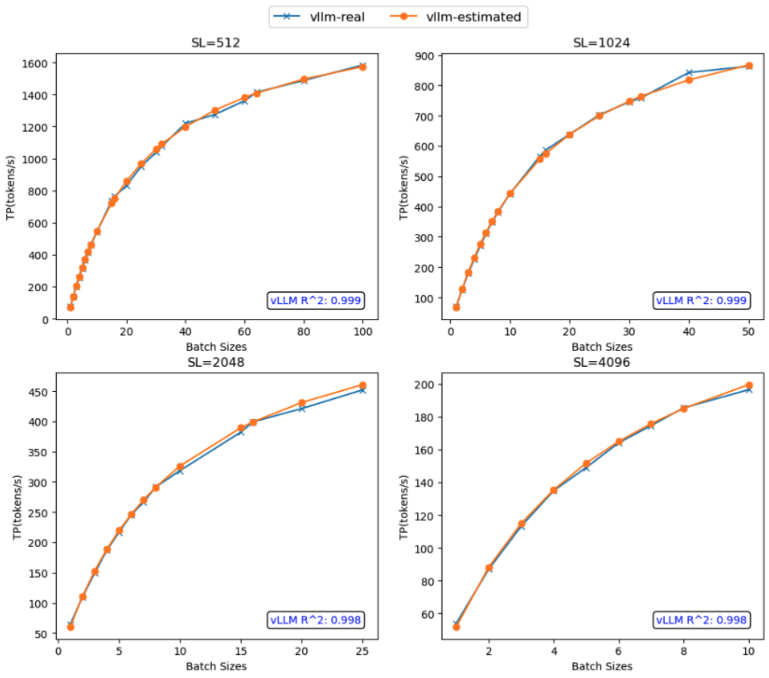


A100-40GB 3g.20gb

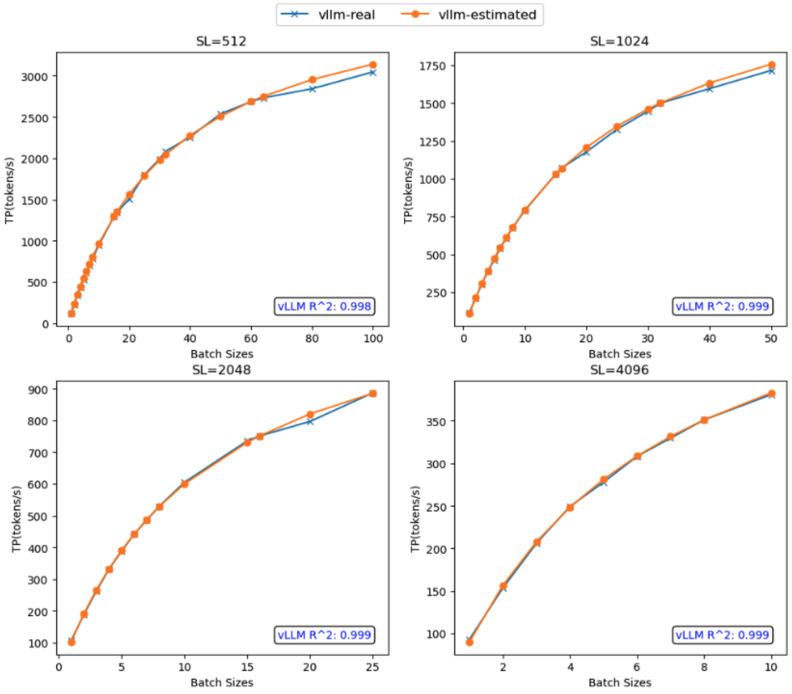
[1] All experiments measured via vLLM-0.6.3

Perf Model Results: Llama3-8B Throughput

- Similarly, measured TP match predicted TP



A100-80GB

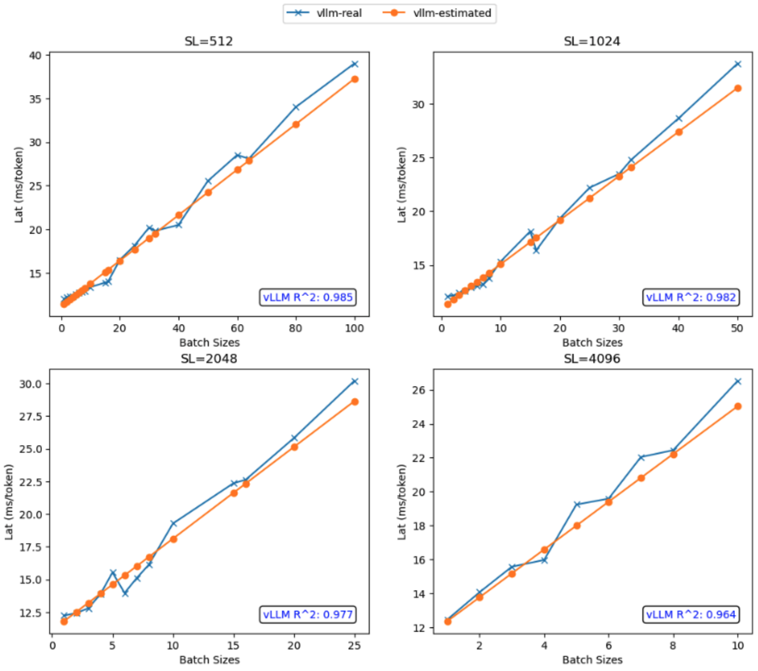


H100-80GB

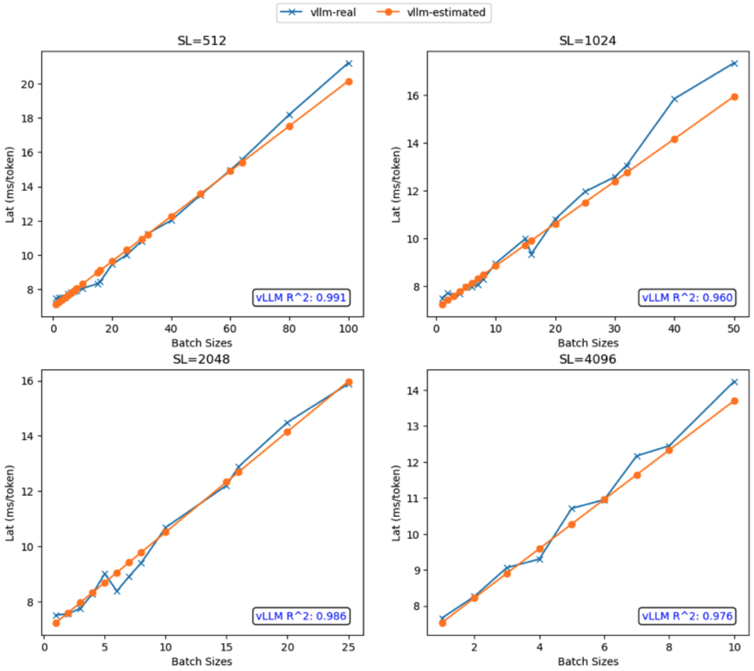
[1] All experiments measured via vLLM-0.6.3

Perf Model Results: Llama3-8B Latency

- Similarly, before ”knee points”, latency linearly increases w/ BS



A100-80GB



H100-80GB

[1] All experiments measured via vLLM-0.6.3

AutoFit

AutoFit Webhook

- Intercept created pod YAMLs
- Extract model characteristics + target load + SLOs (*next slide*)
- Call estimator to decide optimal MIG slice profile
- Patch GPU request in pod spec

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: opt-high
spec:
  containers:
  - resources:
      requests:
        nvidia.com/gpu: 1
...
```


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: opt-high
spec:
  containers:
  - resources:
      requests:
        nvidia.com/mig-2g.10gb: 1
...
```

AutoFit Parser

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: opt-high
spec:
  containers:
  - name: vllm-container
    image: vllm/vllm-openai:v0.6.3
    command: ["bash", "-c"]
    args:
      - |
        vllm serve /model/opt-1.3b --gpu-memory-utilization 0.9 &> vllm.serve.log &
        sleep 60
        python3 /model/workspace/demo/vllm/benchmarks/benchmark_serving.py \
          --model=/model/opt-1.3b \
          --backend=vllm --dataset-name=random --random-input-len=128 \
          --random-output-len=128 --max-concurrency=64 --num-prompts=512
        sleep infinity
    env:
    - name: "TPOT"
      value: "30"
    resources:
      requests:
        nvidia.com/gpu: 1
...
```

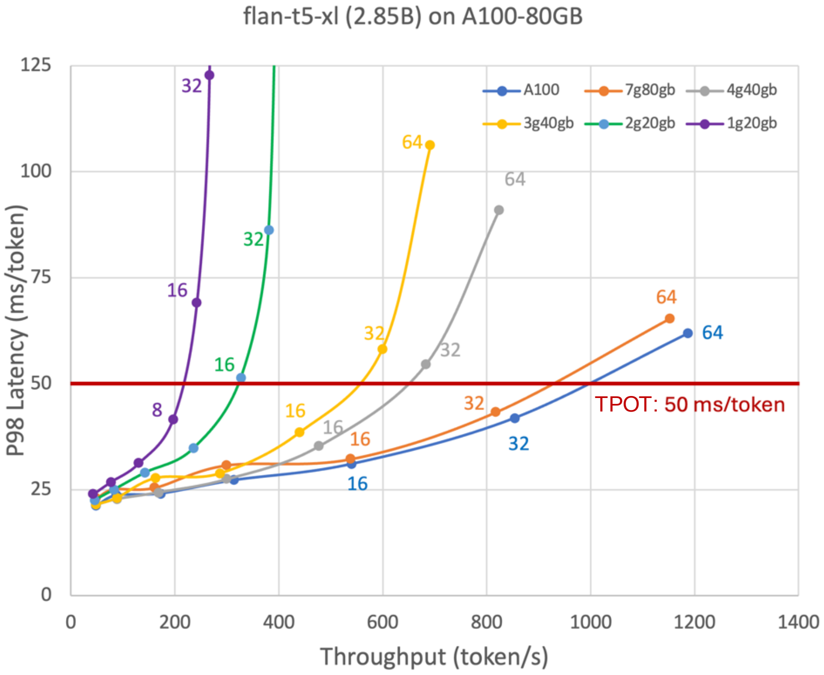
Extract model characteristics

Extract target load characteristics

Extract SLOs

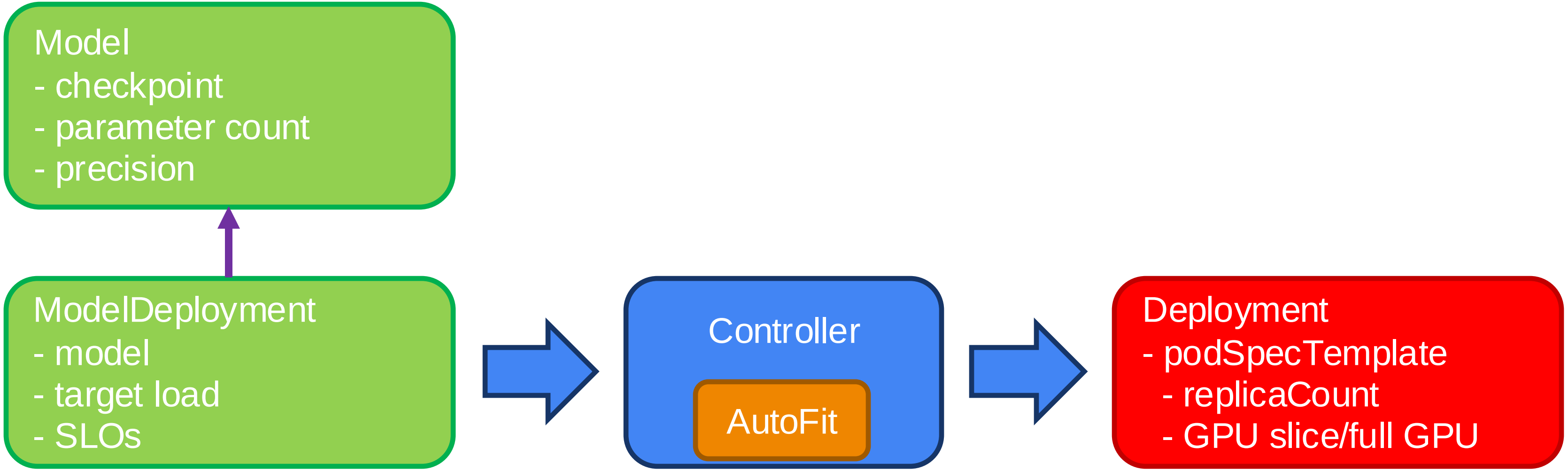
AutoFit Estimator

- For a given TPOT and request load, find the **smallest MIG slice/GPU** based on estimated performance
- Choose **max MIG/GPU** when cannot meet the given TPOT (if request load is too high)
- Example: *TPOT = 50 ms/token*
  - Low Load (=8) → 1g.20gb
  - High Load (=16) → 3g.40gb
  - Very High Load (=64) → full A100 GPU



Model-as-a-Service Architecture

- Two custom resource definitions: *model* and *model deployment*
- Controller generates a *deployment*
  - using AutoFit to scale deployment (both >1 GPU and <1 GPU)



Demo

Demo

- facebook/opt-1.3b
  - Low vs. High Load

- neuralmagic/Mistral-7B-Instruct-v0.3-quantized.w8a16
  - Low vs. Super High Load

- https://youtu.be/NObd7bD7oog

Take Aways

- An analytical model of LLM inference servers
  - latency & throughput = predict(model, batch size, GPU/MIG characteristics)
  - predict accurately predicts the shape but not scale of the performance curve
  - predict can be scaled accurately with just a few profiling data points

- A right-sizing methodology
  - GPU request = recommend(model, expected request load, SLO, available GPUs)

- An open-source project (WIP)
  - AutoFit computes GPU requirements for LLM servers
  - AutoFit adjusts GPU requests for LLM server pods using MIG

KubeCon | CloudNativeCon

North America 2024