Clase 151 — vLLM y TGI: serving de LLMs en producción

Parte: 2 — Deep Learning · Fuente: Kwon et al. (2023) vLLM + HuggingFace TGI docs.

Duración estimada: 80 min.

Clase 151 — vLLM y TGI: serving de LLMs en producción

> *Parte: 2 — Deep Learning · Fuente: Kwon et al. (2023) vLLM + HuggingFace TGI docs. Duración estimada: 80 min.*

Objetivo

Servir LLMs eficientemente en producción con vLLM (Berkeley) o TGI (HuggingFace). Cubrir: PagedAttention, continuous batching, prefill/decode, quantization (AWQ, GPTQ, FP8), structured output (JSON, function calling), streaming, OpenAI-compatible API.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Levantar vLLM serving: python -m vllm.entrypoints.openai.api_server --model X.
- Hacer requests con OpenAI client apuntando a vLLM.
- Aplicar continuous batching y entender ganancia de throughput.
- Servir modelos cuantizados (AWQ, GPTQ, FP8) para reducir VRAM.
- Activar structured outputs con guided_json (JSON schema enforced).

Temas

- KV cache: por qué crece con secuencia.
- PagedAttention (vLLM): page table como OS → no fragmentación.
- Continuous batching: nuevos requests entran sin esperar al batch.
- Prefill (compute KV) vs decode (1 token/step).
- Speculative decoding: predecir N tokens, verificar con modelo grande.
- Quantization: FP8 (H100), AWQ/GPTQ (4-bit weight-only).

Definiciones y características

- vLLM: framework serving open-source. PyTorch-based. ~5-20× throughput vs HF pipeline.
- TGI (Text Generation Inference): HF Hub-integrado. Similar a vLLM.
- PagedAttention: KV cache paginado.
- AWQ / GPTQ: 4-bit weight-only quantization, mantiene calidad ~98 %.
- FP8: 8-bit float native en H100. Mejor que int8 numéricamente.
- Speculative decoding: 2-3× speedup en decode con modelo draft.

Dataset / recursos

- Modelo: Mistral 7B Instruct, Llama 3 8B Instruct, o uno cuantizado AWQ.
- Librerías: vllm, transformers, openai (client).

Ejercicios

1. vLLM básico: python -m vllm.entrypoints.openai.api_server --model mistralai/Mistral-7B-Instruct-v0.2. Cliente OpenAI.
2. Continuous batching benchmark: 100 requests paralelos vs 1 a la vez. Comparar throughput.
3. AWQ quantization: cargar TheBloke/Mistral-7B-Instruct-v0.2-AWQ. VRAM ~5 GB vs 14 GB fp16.
4. Structured JSON output: extra_body={'guided_json': {schema}} → forced JSON valid.
5. TGI: docker run --gpus all ghcr.io/huggingface/text-generation-inference:latest --model-id X.

Homework verificable

Servir Mistral 7B Instruct AWQ con vLLM + cliente OpenAI:

1. Levantar server vLLM.
2. 50 prompts batch en paralelo desde cliente.
3. Medir throughput (tokens/sec total).
4. Comparar contra transformers.pipeline generate.
5. Activar guided_json para una tarea específica.

Criterio de aceptación: vLLM ≥ 5× throughput vs HF pipeline; structured output válido al 100 %.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OOM al cargar	Modelo + KV cache no entra. Fix: quantizat
Throughput bajo en una sola request	Con batch=1, vLLM es similar a HF. Fix: pa
LoRA adapter no carga	Fix: vLLM soporta LoRA con --enable-lora -
Structured output formato inválido	guided_json schema mal. Fix: JSON Schema v
TGI lento	Versión vieja. Fix: usar imagen latest.

Preguntas frecuentes

vLLM vs TGI?

vLLM ligeramente más rápido en benchmarks; mejor LoRA support. TGI más integrado con HF Hub. Ambos buenos.

Ollama / llama.cpp?

llama.cpp / Ollama: CPU-friendly, GGUF format. Para desktop, local, low traffic. vLLM/TGI para servers con GPU.

Estructura JSON garantizada?

Sí con guided_json (vLLM uses outlines library). Mucho más confiable que prompt engineering.

Speculative decoding?

--speculative-model X --num-speculative-tokens 5. 2-3× decode speedup con calidad idéntica.

Multi-GPU?

--tensor-parallel-size N para shard del modelo en N GPUs.

Referencias

- Kwon et al. (2023), Efficient Memory Management for LLM Serving with PagedAttention, SOSP.
- vLLM docs.
- TGI docs.
- Outlines (structured output).

Siguiente clase

Clase 152 — RAG básico y embeddings (+ hybrid search, re-ranking, MCP)

Apéndice: notebook (primer bloque)

Simulamos continuous batching y PagedAttention con asyncio + colas. La API real de vLLM se muestra en markdown.

```
import numpy as np, asyncio, time, random
from collections import deque
random.seed(42); np.random.seed(42)
```

Archivos complementarios

- notebook.ipynb