IA local para PyMEs · Módulo 4

Inferencia local

Del runtime de escritorio al servidor con concurrencia. Medir antes de opinar.

Qué se construye en este módulo

Runtime de un usuario contra servidor con concurrencia

Elegir runtime según usuarios simultáneos

Cuantización: el costo real en calidad

Medir la pérdida con un set fijo

Medir: tokens/s, TTFT y prefill

Construir un banco de pruebas reproducible

Qué se construye en este módulo

Por qué «funciona en mi máquina» muere con cinco usuarios

Hacer una prueba de carga con cinco usuarios

4.1 · Runtime de un usuario contra servidor con concurrencia

Runtime de un usuario contra servidor con concurrencia

Todo lo que sirve un modelo en local cae en una de dos familias. Confundirlas es la causa número uno de «funcionaba en la demo y se cayó en la oficina».

Las dos exponen una **API compatible con OpenAI** (`/v1/chat/completions`). Esa es la interfaz neutral del curso: **el asistente habla con esa API y no sabe qué hay detrás.** Cambiar de runtime a servidor —o de modelo— no toca el código del asistente.

En una tabla

Perfil	Qué pasa
A	Solo runtime de escritorio; un usuario a la vez; lote nocturno
B	Runtime cómodo; servidor con concurrencia **cabe** con modelos pequeños y contexto corto
C	Servidor con concurrencia y caché de prefijos; 10+ usuarios

El comando, copiable

```
cd ~/labs/el-tornillo && mkdir -p labs/04-01 && cd labs/04-01
# 1) runtime de escritorio (perfil A/B) — un comando, API en :8080
llama-server -m <modelo>.gguf -c 8192 --port 8080
# 2) servidor con concurrencia (perfil B/C, CUDA) — contenedor, API en :8000
docker run --gpus all -p 8000:8000 <imagen-del-servidor> --model <modelo> --max-
    model-len 8192
# 3) la misma petición a los dos
for p in 8080 8000; do curl -s localhost:$p/v1/chat/completions -H 'content-type:
    application/json' \
  -d '{"model":"local","messages":[{"role":"user","content":"Diga hola en una línea
    ."}]}' | jq -r '.choices[0].message.content'; done
```

Si no corre desde cero con un comando, no entra al curso.

Cuándo NO usar esto

Runtime de un usuario contra servidor con concurrencia

- No use un runtime de escritorio como servidor para más de dos usuarios: se serializa.
- No instale un servidor con concurrencia en un portátil sin GPU: no corre, o corre peor que el runtime.
- No ate el asistente a la API propia de un runtime (endpoints no estándar): en el momento de cambiar, se reescribe.

Qué se degrada en cada perfil

A · sin GPU

Solo runtime de escritorio; un usuario a la vez; lote nocturno

B · 8–16 GB VRAM

Runtime cómodo; servidor con concurrencia **cabe** con modelos pequeños y contexto corto

C · 24 GB+

Servidor con concurrencia y caché de prefijos; 10+ usuarios

4.2 · Cuantización: el costo real en calidad

Cuantización: el costo real en calidad

En la lección 0.3 se dijo que la cuantización cuesta calidad «en tareas finas». Aquí se mide con la tarea de la ferretería: **extraer campos de facturas**. Un método, tres cuantizaciones, un número por cada una.

30 facturas de proveedor (reales, anonimizadas) con sus campos anotados a mano: número, fecha, proveedor, NIT, subtotal, IVA, total. `labs/04-02/set/` con un `esperado.json`.

En una tabla

Si...	Entonces
Q4 pierde menos de 1 punto	Q4: cabe más contexto y más usuarios
Q4 pierde 3–6 puntos	Q6 o Q8, aunque quepan menos usuarios; o un modelo m queño en Q8
Q8 tampoco llega al objetivo	el problema no es la cuantización: es el prompt, el esque OCR (módulos 5 y 7)

Cuándo NO usar esto

Cuantización: el costo real en calidad

- No mida con 5 facturas: el ruido es mayor que la diferencia. Mínimo 30; mejor 100.
- No compare cuantizaciones de modelos distintos: eso es otra pregunta (¿cuál modelo?), con el mismo método.
- No cuantice por debajo de Q4 para tareas de cifras.

Qué se degrada en cada perfil

A · sin GPU

Solo Q4 (y Q6 con paciencia); la medición tarda horas, pero es la misma

B · 8–16 GB VRAM

Q6/Q8 en modelos medianos; se mide en minutos

C · 24 GB+

Q8 o F16 en modelos medianos; se puede medir contra la referencia sin cuantizar

4.3 · Medir: tokens/s, TTFT y prefill

Medir: tokens/s, TTFT y prefill

Tres números por corrida, en un JSON, con fecha, versión del runtime, modelo, cuantización y hardware. Sin ese JSON no hubo medición: hubo una impresión.

Lo que debe ver: un JSON con `ttft_s` y `decode_tok_s`. Se corre **tres veces** y se guarda la mediana. Se repite cada vez que cambie algo: modelo, cuantización, runtime, driver, contexto.

En una tabla

Número	Qué es	Cómo se obtiene con la API es tándar
TTFT	ms hasta el primer token	tiempo desde el envío has el primer `data:` con contenic (streaming)
tokens/s de decode	velocidad de escritura	tokens de la respuesta ÷ (tiemp total – TTFT)
tokens/s de prefill	velocidad de lectura	tokens de entrada ÷ TTFT (apro ximación válida si la entrada e larga)

Cuándo NO usar esto

Medir: tokens/s, TTFT y prefill

- No mida con la GPU compartida con otra cosa (el navegador con video, otro modelo cargado).
- No compare TTFT de entradas de distinto tamaño.
- No publique un número sin su JSON.

Qué se degrada en cada perfil

A · sin GPU

5–15 tok/s

B · 8–16 GB VRAM

30–80 tok/s

C · 24 GB+

80–200 tok/s

4.4 · Por qué «funciona en mi máquina» muere con cinco usuarios

Por qué «funciona en mi máquina» muere con cinco usuarios

Un asistente que responde en 3 segundos a **una** persona puede tardar 40 con **cinco**. El cliente no compra «funciona»; compra «funciona el martes a las 10, cuando los tres están usándolo». Este laboratorio lo simula.

Lo que debe ver `[VOLÁTIL — verificado: 2026-09]`, orden de magnitud en perfil B con un 8B Q4:

En una tabla

	1 usuario	5 usuarios
Runtime de escritorio	TTFT ~3 s	usuario 4: ~15 s; usuario 5: ~ (cola)
Servidor con batching	TTFT ~3 s	todos entre 3,5 y 5 s

El comando, copiable

```
python3 carga.py http://localhost:8080/v1/chat/completions 1
python3 carga.py http://localhost:8080/v1/chat/completions 5
python3 carga.py http://localhost:8000/v1/chat/completions 5    # el servidor con
    concurrencia
```

Si no corre desde cero con un comando, no entra al curso.

Cuándo NO usar esto

Por qué «funciona en mi máquina» muere con cinco usuarios

- No pruebe carga contra el equipo del cliente en horario laboral.
- No extrapole de 5 a 50 usuarios: a partir de cierto punto la memoria, no el cómputo, es el techo.

Qué se degrada en cada perfil

A · sin GPU

Un usuario. La prueba de 5 sirve para **demostrarle al cliente** por qué hace falta la fase 2

B · 8–16 GB VRAM

3–5 usuarios con batching y contexto corto

C · 24 GB+

10+

Entregables del módulo

- los dos comandos de arranque en `labs/04-01/arranque.md` con la versión y la fecha.
- `resultados-*.json` y una tabla en `labs/04-02/decision.md`.
- `bench.py`, tres JSON, y `labs/04-03/historial.md` con una fila por corrida.
- `carga.py` y `labs/04-04/carga.md` con las tres tablas.

Del runtime de escritorio al servidor con concurrencia.