> **LO ESENCIAL · Módulo 1 · Elección de hardware**
>
> Los tres números que deciden todo, y cómo comprar —para uno mismo y para un cliente— sin casarse con una marca.

1.1 · Los tres números: ancho de banda, capacidad y FLOPS

> Objetivos
>
> - Estimar tokens/s y TTFT en papel
> - Descartar un equipo antes de comprarlo

Antes de mirar marcas, tres números

Cualquier equipo para inferencia local se describe con tres números. Con ellos se puede predecir en papel, con un margen de ±30 %, cómo va a rendir **antes de comprarlo**. Sin ellos se compra por fe.

Número	Unidad	Qué decide	Dónde se lee
Capacidad de memoria	GB	si el modelo + su contexto **caben**	ficha técnica: VRAM (GPU) o memoria unificada
Ancho de banda de memoria	GB/s	la velocidad de **decode** (tokens/s)	ficha técnica: *memory bandwidth*
Cómputo	TFLOPS (FP16/BF16)	la velocidad de **prefill** (TTFT)	ficha técnica: *FP16 performance*

El orden importa: primero capacidad (si no cabe, nada más importa), luego ancho de banda (la sensación de velocidad), y por último cómputo (que manda en documentos largos, lección siguiente).

Capacidad: ¿cabe?

```
memoria necesaria ≈ pesos + caché de contexto + margen del runtime
pesos             = parámetros × bytes/parámetro     (lección 0.3)
caché de contexto ≈ depende del modelo; orden de ~0,10–0,15 GB por cada 1 000 tokens en un 8B
margen            ≈ ~0,5–1 GB
```

Ejemplo: un 8B en Q4 (~4,9 GB) con 16k de contexto (~2 GB) y margen → **~8 GB**. Cabe justo en una tarjeta de 8 GB, sin dejar nada para un segundo usuario. En 12 GB respira. En 6 GB, no corre o corre partido entre GPU y RAM (y entonces manda el ancho de banda de la RAM, que es 5–10 veces menor).

Ancho de banda: tokens por segundo

Para generar **un** token, el modelo tiene que leer **todos** sus pesos una vez. Por eso:

```
tokens/s (decode, un usuario) ≈ ancho de banda (GB/s) ÷ tamaño de los pesos (GB)
```

Equipo (clase) `[VOLÁTIL — verificado: 2026-09]`	Ancho de banda	8B Q4 (4,9 GB)
Portátil sin GPU, RAM DDR5 doble canal	~80 GB/s	~16 tok/s
GPU de consumo gama media (8–16 GB GDDR6)	~300–500 GB/s	~60–100 tok/s
Estación Apple Silicon gama alta (memoria unificada)	~400–800 GB/s	~80–160 tok/s
GPU de consumo tope de gama (24 GB GDDR6X/7)	~900–1 000 GB/s	~180–200 tok/s

La fórmula es un techo teórico; en la práctica se obtiene el 60–80 %. **Pero el orden de magnitud no falla**: una CPU no va a dar 60 tok/s con un 8B, por mucho software que se le ponga.

> Nota Para un humano leyendo, 15–20 tok/s ya parece «rápido». Para un agente que hace diez pasos por tarea, 15 tok/s significa minutos por tarea. El mismo número es suficiente o insuficiente según el uso.

Cómputo: el tiempo hasta la primera palabra

Antes de escribir, el modelo tiene que **procesar todo el contexto** (prefill). Eso no es leer pesos una vez: son multiplicaciones de matrices proporcionales a `tokens × parámetros`. Manda el cómputo:

```
tiempo de prefill ≈ (2 × parámetros × tokens de entrada) ÷ FLOPS efectivos
```

Ejemplo: 8B, 6 000 tokens de entrada (un contrato de 12 páginas):

Equipo (clase)	FLOPS efectivos FP16	Prefill de 6k tokens
CPU portátil	~1 TFLOPS	~100 s
GPU de consumo gama media	~30–60 TFLOPS	~2–3 s
GPU tope de gama	~100+ TFLOPS	< 1 s

Cien segundos hasta la primera palabra es un sistema que **nadie usa**. Ahí es donde el perfil A se rompe para cargas documentales, y por qué la lección siguiente es la más importante del módulo.

> PROCEDIMIENTO · **Laboratorio 1.1 · la calculadora**
>
> ```
> cd ~/labs/el-tornillo && mkdir -p labs/01-01 && cd labs/01-01
> cat > calculadora.csv <<'EOF'
> equipo,memoria_gb,ancho_banda_gbs,tflops_fp16,modelo_gb,contexto_tokens
> portatil-sin-gpu,16,80,1,4.9,8000
> gpu-media-12gb,12,360,40,4.9,16000
> ```

```
servidor-24gb,24,1000,120,8.5,32000
EOF
python3 - <<'PY'
import csv
for r in csv.DictReader(open("calculadora.csv")):
    m=float(r["modelo_gb"]); ctx=int(r["contexto_tokens"])
    necesita = m + 0.12*ctx/1000 + 0.8
    cabe = "sí" if necesita <= float(r["memoria_gb"]) else "NO"
    tps = float(r["ancho_banda_gbs"])/m*0.7
    prefill = 2*8e9*6000/(float(r["tflops_fp16"])*1e12*0.5)
    print(f'{r["equipo"]:18s} necesita {necesita:5.1f} GB → cabe: {cabe:3s} | ~{tps:5.0f}
     tok/s | prefill 6k: {prefill:6.1f} s')
PY
```

Lo que debe ver: tres líneas. El portátil «cabe» pero con prefill de más de un minuto; el servidor cabe con todo y responde en segundos. **Ese es el resultado que se le muestra al cliente antes de que compre.**

Entregable: `calculadora.csv` con los tres equipos que usted tenga a mano o esté considerando.

Qué se degrada en cada perfil

A — Cabe lo pequeño; el prefill de documentos largos es inviable para uso diario

B — Modelos medianos cuantizados; contexto limitado por la caché; prefill aceptable

C — Modelos grandes o medianos en Q8 con contexto amplio; prefill en segundos

Cuándo NO usar esto **Los tres números: ancho de banda, capacidad y FLOPS**

- No use la calculadora para comparar **software** (dos runtimes en el mismo equipo): eso se mide.
- No compre por TFLOPS si la carga es chat corto: ahí manda el ancho de banda.
- No confíe en la cifra de memoria «compartida» de un portátil: la GPU integrada usa RAM lenta.

PRÁCTICA · **Práctica**

1. Un cliente ofrece dos opciones: 16 GB a 400 GB/s o 12 GB a 900 GB/s. ¿Cuál para un 14B Q4 con 8k de contexto? ¿Y para un 8B Q8 con 32k?
2. Estime el prefill de una remisión de 2 páginas (~1 000 tokens) en el perfil A. ¿Es usable?

3. Explique por qué duplicar la RAM de un portátil no duplica los tokens/s.

Referencias

1. Documentación de llama.cpp sobre rendimiento y `llama-bench`: https://github.com/ggml-org/llama.cpp `[VOLÁTIL — verificado: 2026-09]`
2. Pope, R. et al. (2022). *Efficiently Scaling Transformer Inference.* arXiv:2211.05102 — el análisis de memoria y cómputo del que salen las dos fórmulas.
3. Kwon, W. et al. (2023). *Efficient Memory Management for Large Language Model Serving with PagedAttention.* arXiv:2309.06180 — por qué la caché de contexto es el segundo consumidor de memoria.

1.2 · Por qué el prefill domina en cargas documentales y agénticas

> Objetivos
>
> - Medir prefill y decode por separado
> - Explicar por qué RAG y agentes son prefill-bound

La lección más importante del módulo

Casi todas las comparativas que circulan miden **tokens por segundo de salida**. Es el número equivocado para una PyME que quiere leer sus documentos. En cargas documentales y agénticas, lo que domina es el **prefill**: el tiempo de procesar la entrada antes de escribir la primera palabra.

Dos razones:

1. **La entrada es mucho más larga que la salida.** Un contrato de 6 000 tokens produce un resumen de 200. La proporción 30:1 es normal.
2. **Los agentes releen.** En cada paso del bucle, el contexto completo (instrucciones + historia + resultados de herramientas) vuelve a entrar. Diez pasos = diez prefills.

Medirlo, no suponerlo

Los runtimes locales reportan las dos velocidades por separado. Con `llama-bench` de llama.cpp `[VOLÁTIL — verificado: 2026-09]` (el principio es idéntico en cualquier runtime que exponga

métricas):

```
cd ~/labs/el-tornillo && mkdir -p labs/01-02 && cd labs/01-02
# -p = tokens de prefill a medir, -n = tokens de decode a medir
llama-bench -m <modelo>.gguf -p 512,2048,6144 -n 128 -o json > bench.json
python3 - <<'PY'
import json
for r in json.load(open("bench.json")):
    tipo = "prefill" if r["n_prompt"] > 0 else "decode "
    n = r["n_prompt"] or r["n_gen"]
    print(f'{tipo} {n:5d} tokens → {r["avg_ts"]:8.1f} tok/s  ({n/r["avg_ts"]:5.1f} s)')
PY
```

Lo que debe ver: el prefill en tok/s es **mucho mayor** que el decode (es paralelo), pero al multiplicarlo por 6 144 tokens el tiempo total puede ser de segundos en GPU y de **minutos en CPU**. Anote los dos tiempos: son la línea base del proyecto.

Qué significa para el diseño

Carga	Manda	Diseño
Chat corto (preguntas sueltas)	decode	ancho de banda; casi cualquier equipo
Resumir, extraer, responder con RAG	**prefill**	cómputo; contexto lo más corto posible
Agentes de varios pasos	**prefill** × **pasos**	cómputo **y** caché de prefijos

De aquí salen dos reglas que atraviesan el curso:

- **Contexto corto es dinero.** Cada token de instrucciones innecesarias se paga en cada paso. Módulo 5.
- **Caché de prefijos.** Si las instrucciones y los documentos base no cambian entre pasos, el servidor puede reutilizar su prefill. Los servidores con concurrencia lo hacen; los runtimes de escritorio, a medias. Módulo 4.

> PROCEDIMIENTO · **Laboratorio 1.2 · prefill contra decode en su equipo**
>
> Además del banco de arriba, mida el efecto de la caché de prefijos con dos llamadas seguidas que compartan las primeras 4 000 tokens (mismo documento, distinta pregunta). Con un runtime que cachea, la segunda llamada tiene un TTFT **muy** inferior.
>
> ```
> # ttft.py — tiempo hasta el primer token con la API compatible OpenAI del runtime local
> import time, requests, json
> URL = "http://localhost:8080/v1/chat/completions"     # ajuste al puerto de su runtime
> doc = open("contrato.txt").read()
> def ttft(pregunta):
>     t0 = time.perf_counter()
>     with requests.post(URL, json={"model":"local","stream":True,
> ```

```
        "messages":[{"role":"system","content":"Responda solo con datos del documento."},
                    {"role":"user","content":doc+"\n\nPregunta: "+pregunta}]}, stream=True
    ) as r:
        for linea in r.iter_lines():
            if linea.startswith(b"data:") and b'"content"' in linea:
                return time.perf_counter()-t0
print("1.ª llamada TTFT:", round(ttft("¿Cuál es la fecha de firma?"),2), "s")
print("2.ª llamada TTFT:", round(ttft("¿Quién es el proveedor?"),2), "s")
```

Entregable: `bench.json` y las dos cifras de TTFT. Commit: «Capa 1: línea base de rendimiento».

Qué se degrada en cada perfil

A — Prefill de documentos largos en minutos: solo sirve para lotes nocturnos

B — Prefill en segundos; la caché de prefijos hace usable un agente de pocos pasos

C — Prefill sub-segundo; agentes de muchos pasos y varios usuarios

Cuándo NO usar esto **Por qué el prefill domina en cargas documentales y agénticas**

- No optimice prefill si la carga real es chat corto: mida primero qué hacen los usuarios.
- No mida con un prompt de 50 tokens y extrapole: el prefill no es lineal en todos los runtimes.

PRÁCTICA · **Práctica**

1. Un agente hace 8 pasos con 5 000 tokens de contexto cada uno. En un equipo con prefill de 400 tok/s, ¿cuánto tarda solo en prefill? ¿Y con caché de prefijos que ahorre el 80 %?
2. ¿Por qué un servidor con **la mitad** de tokens/s de decode puede sentirse más rápido en RAG?

Referencias

1. llama.cpp — `llama-bench` y métricas de prefill/decode: https://github.com/ggml-org/llama.cpp
2. Kwon, W. et al. (2023). PagedAttention — arXiv:2309.06180 (caché y batching).
3. Zheng, L. et al. (2023). *SGLang: Efficient Execution of Structured Language Model Programs*. arXiv:2312.07104 — RadixAttention: reutilización de prefijos entre llamadas.

1.3 · Comparación razonada de cinco topologías

> Objetivos
>
> - Decidir entre GPU de consumo, memoria unificada CUDA, Apple Silicon, mini PC y SBC
> - Asignar a cada uno su rol en la topología

Cinco clases de equipo, no cinco marcas

Las marcas cambian cada seis meses; las **clases** no. Cada una tiene un rol en la topología y un «cuándo no». Todo lo que sigue va con `[VOLÁTIL — verificado: 2026-09]` en cifras; el criterio es estable.

1 · GPU discreta de consumo

Una tarjeta gráfica de videojuegos en un PC normal. 8–24 GB de VRAM, 300–1 000 GB/s.

- **Cuándo sí:** es la mejor relación tokens/s por peso en el perfil B y C. Software maduro.
- **Cuándo no:** más de 24 GB exige dos tarjetas (complejidad, calor, ruido, fuente de 1 000 W). Ruido y consumo en una oficina pequeña. Modelos > 30B no caben con calidad.
- **Rol: nodo de cómputo**. Nada más corre ahí.

2 · Appliance de memoria unificada con CUDA

Equipos pequeños con CPU+GPU del mismo fabricante compartiendo 64–128 GB de memoria y el ecosistema CUDA. Ancho de banda medio (~250–300 GB/s), mucha capacidad.

- **Cuándo sí:** modelos grandes (70B en Q4) en una caja silenciosa; desarrollo con el mismo stack que un servidor; poco consumo.
- **Cuándo no:** decode más lento que una GPU tope de gama (menos ancho de banda); precio alto por token/s; producto joven —repuestos y garantía locales inciertos.
- **Rol:** nodo de cómputo para **modelos grandes con pocos usuarios**, o estación del implementador.

3 · Estación Apple Silicon

Memoria unificada de 32–192 GB con ancho de banda alto en las gamas altas, silencio, bajo consumo, Metal en vez de CUDA.

- **Cuándo sí:** un solo puesto (el gerente, el implementador) con modelos grandes; oficina donde el ruido importa; ya hay Macs.
- **Cuándo no:** servidor multiusuario (el ecosistema de servidores con concurrencia es CUDA); precio de la memoria alta; no se amplía.
- **Rol:** estación de trabajo o nodo de cómputo de **un** usuario.

4 · Mini PC (x86 de bajo consumo)

Cajas de 10–30 W con CPU de portátil, 16–64 GB de RAM, sin GPU útil para inferencia.

- **Cuándo sí: nodo de servicios**: proxy, SSO, base de datos, colas, RAG (el índice, no el modelo), automatizaciones, trazas. Silencioso, barato, siempre encendido.
- **Cuándo no:** inferencia de cualquier cosa que no sea diminuta. Embeddings y reranking pequeños sí; un 8B a 5 tok/s no es un servicio.
- **Rol:** el **cerebro operativo** que no piensa: orquesta, guarda, autentica.

5 · Computador de placa única (SBC)

Placas de 5–15 W (clase Raspberry Pi y similares), 4–16 GB de RAM.

- **Cuándo sí:** sensores, un canario, un nodo de la malla, un espejo de backups fuera de la oficina (casa del gerente).
- **Cuándo no:** cualquier inferencia; cualquier servicio con más de un usuario.
- **Rol: periferia**. Barato de tener encendido en otro sitio.

La tabla que se le muestra al cliente

Clase	Cómputo	Servicios	Almacena-miento	Periferia	Ruido	Cons
GPU de consumo	●	—	—	—	alto	300–6
Appliance CUDA unificado	● (grandes)	—	—	—	bajo	100–2
Apple Silicon	● (1 usuario)	○	—	—	mínimo	30–15
Mini PC	—	●	○	—	mínimo	10–30
SBC	—	—	○ (espejo)	●	ninguno	5–15

● rol natural · ○ posible · — no

Cuándo NO usar esto **Comparación razonada de cinco topologías**

- No diseñe **todo en una caja**. El nodo de cómputo se apaga para actualizar drivers; el proxy y la base de datos no pueden apagarse con él. Lección siguiente.
- No compre la clase 2 o 3 «porque tiene mucha memoria» para un caso de 5 usuarios con un 8B: una GPU de consumo de 16 GB rinde más por menos.
- No ponga un NAS a computar (módulo 2).

PRÁCTICA · **Práctica**

1. Ferretería, 3 usuarios, documentos de 1–12 páginas, presupuesto medio. Asigne una clase a cada rol (cómputo, servicios, almacenamiento, periferia) y justifique con los tres números.
2. Un cliente insiste en un Mac Studio como servidor para 8 usuarios. Escriba en cinco líneas qué se degrada y qué alternativa propone.

Referencias

1. Fichas técnicas de los fabricantes: los tres números se leen allí, no en reseñas.
2. llama.cpp — backends soportados (CUDA, Metal, CPU) y sus límites: https://github.com/ggml-org/llama.cpp
3. Módulo 4 de este curso — dónde el ecosistema de servidores con concurrencia sí exige CUDA.

1.4 · Separación de nodos y compra para un cliente

> Objetivos
>
> - Aplicar la regla cómputo ≠ servicios ≠ almacenamiento
> - Especificar hardware que no dependa del implementador

La regla de separación

> Nota **Nodo de cómputo ≠ nodo de servicios ≠ almacenamiento.**

Tres funciones, tres cajas (o al menos tres máquinas virtuales bien separadas, si el presupuesto obliga). Razones concretas:

Si se juntan…	Pasa esto
cómputo + servicios	actualizar el driver de la GPU tumba el proxy, el SSO y la base de datos: **nadie entra**
cómputo + almacenamiento	la GPU calienta los discos; un reinicio por un modelo colgado corta el acceso a los archivos
servicios + almacenamiento	aceptable en PyMEs pequeñas si el NAS tiene contenedores — pero el NAS **no** debe computar

Orden de compra por presupuesto creciente

El error típico es comprar la GPU primero. La GPU es lo que **más cambia de precio** y lo que **menos dura** en el diseño. Se compra al final.

Escalón	Qué se compra	Qué permite ya
1 · servicios	mini PC (16–32 GB RAM, 2 SSD)	malla, proxy, SSO, trazas, índice RAG, automatizaciones. Inferencia pequeña o en el perfil A
2 · almacenamiento	NAS de 2–4 bahías + discos + copia externa	documentos, backups 3-2-1, restauración probada
3 · cómputo	GPU de consumo 16–24 GB en un PC dedicado	modelos medianos, 3–10 usuarios, prefill en segundos
4 · ampliar	segunda GPU, o appliance para modelos grandes	solo si las trazas del escalón 3 muestran cola

Con los escalones 1 y 2 el sistema **ya funciona** (lento, con modelos pequeños, o en lote nocturno). Eso permite vender diagnóstico y primeras automatizaciones sin que el cliente ponga la GPU sobre la mesa.

Comprar PARA UN CLIENTE: otros criterios

Cuando el equipo es propio, se optimiza tokens/s por peso. Cuando es del cliente, se optimiza **que siga funcionando sin usted**:

Criterio	Por qué
Garantía local y repuestos	una GPU importada sin garantía en Colombia es un riesgo que asume el cliente sin saberlo
Silencio y consumo	va a estar en una oficina, no en un cuarto técnico
Componentes estándar	que cualquier técnico local pueda cambiar la fuente o el disco
Factura a nombre del cliente	el activo es suyo; el implementador no debe ser dueño de nada
Sin dependencia del implementador	contraseñas de BIOS, cuentas de administración y llaves: **entregadas** y documentadas (módulo 14)
UPS	los cortes de luz en Colombia no son hipotéticos: un NAS sin UPS pierde datos

ATENCIÓN · **Ojo**

El equipo más barato para usted puede ser el más caro para el cliente. Un PC armado a mano ahorra un 20 % y le cuesta al cliente cada visita técnica futura.

PROCEDIMIENTO · **Laboratorio 1.4 · la especificación de la ferretería**

Escriba `labs/01-04/especificacion.md` con front-matter y tres secciones: **nodo de servicios**, **almacenamiento**, **nodo de cómputo**. Para cada uno: clase, los tres números, presupuesto en COP (rango), qué permite en cada perfil (A/B/C), garantía y quién lo compra.

```
cd ~/labs/el-tornillo && mkdir -p labs/01-04
# … escribir especificacion.md …
git add labs/01-04 && git commit -qm "Capa 1: especificación de hardware" && git log --
    oneline -1
```

Entregable: la especificación, con el escalón 1+2 marcado como «fase 1» y el 3 como «fase 2».

Qué se degrada en cada perfil

A — —

B — 3–5 usuarios

C — 10+ usuarios, modelos grandes

Cuándo NO usar esto **Separación de nodos y compra para un cliente**

- No separe en tres cajas físicas si el cliente tiene **un** empleado y **un** caso de uso: dos máquinas virtuales en un PC con GPU bastan; la regla de separación se cumple igual.

- No compre la GPU antes de tener trazas que digan **cuánto** se usa.

PRÁCTICA · **Práctica**

1. Un cliente ofrece un servidor viejo con 2 CPU y 128 GB de RAM «para la IA». ¿Qué rol le da?
2. Escriba la lista de lo que se **entrega** al cliente el día uno (cuentas, llaves, garantías).

Referencias

1. Módulo 3 (red) y módulo 14 (entrega) de este curso.
2. Ley 1581 de 2012 y Decreto 1377 de 2013 — dónde viven los datos también es una decisión legal (módulo 2.3).