

LO ESENCIAL · Módulo 0 · Fundamentos y vocabulario

Las palabras con las que se habla el resto del curso, definidas por lo que hacen. Y un glosario con símiles para quien llega de cero.

0.1 · Glosario con símiles: para entender de qué se habla

Objetivos

- Explicar cada término del curso con un símil cotidiano
- Distinguir modelo, agente, herramienta y RAG sin jerga

Para qué sirve esta página

Este curso usa unas cuarenta palabras que en las noticias suenan a magia y en la práctica son cosas concretas y medibles. Aquí están todas, cada una con un **símil de la vida diaria** y, al lado, la advertencia de **dónde se rompe el símil** —porque todo símil se rompe, y saber dónde es la mitad de entender el término.

No hace falta memorizar nada. Se lee una vez y se vuelve cada vez que una palabra estorbe.

Nota Regla del curso: si un término no se puede medir o ejecutar, no se usa. Cada entrada de abajo termina diciendo **cómo se ve en la práctica**.

El modelo y sus partes

Término	Símil	Dónde se rompe el símil	Cómo se ve en la práctica
Modelo	Un empleado nuevo que leyó toda la biblioteca pública pero nunca ha visto la empresa	El empleado aprende con el tiempo; el modelo, tal como se descarga, no aprende nada de lo que usted le diga	Un archivo de varios gigas que se carga en memoria y responde texto
Pesos / parámetros	Los miles de millones de «perillas» que quedaron ajustadas cuando lo entrenaron	Nadie ajusta perillas a mano; se ajustan solas con datos	El número que acompaña al nombre: «8B» son ocho mil millones de parámetros
Entrenamiento	Los años de lectura del empleado, hechos por otro, antes de llegar	Cuesta millones y no se repite en una PyME ; lo que sí se hace es «especializar» (módulo 13)	Ya viene hecho. Usted descarga el resultado
Inferencia	Hacerle una pregunta al empleado y esperar la respuesta	—	Cada vez que el modelo responde. Es lo que se paga en hardware
Cuantización	Guardar una foto en JPEG en vez de RAW: pesa la cuarta parte y casi no se nota	«Casi» no es «nunca»: en tareas finas la pérdida se nota, y hay que medirla (módulo 4)	El sufijo del archivo: Q8, Q4... menos bits, menos memoria, algo menos de calidad
Token	Las sílabas con las que el modelo lee: no lee palabras ni letras, lee pedazos	Las sílabas del español no coinciden con los tokens; una palabra rara puede ser 5 tokens	Se cuentan con un programa. 1.000 palabras en español ≈ 1.400–1.800 tokens
Contexto / ventana	La mesa de trabajo del empleado: lo que cabe en la mesa es lo único que ve mientras responde	La mesa real se puede ampliar; la ventana tiene un tope duro, y llenarla cuesta tiempo y memoria	Un número: 8k, 32k, 128k tokens. Todo lo que se le da al modelo compite por ese espacio
Prefill	Leer todo lo que hay sobre la mesa antes de empezar a contestar	—	El tiempo hasta la primera palabra. Manda en documentos largos
Decode	Dictar la respuesta palabra por palabra	—	Tokens por segundo. Manda en respuestas largas
Alucinación	El empleado que, antes que decir «no sé», inventa con toda seguridad	No es mentira ni malicia; es la forma en que funciona: siempre produce el texto más probable	Se combate dándole las fuentes (RAG) y validando salidas (esquemas)

Memoria y recuperación

Término	Símil	Dónde se rompe el símil	Cómo se ve en la práctica
Embedding	Convertir un texto en coordenadas en un mapa : textos parecidos quedan cerca	En un mapa hay 2 dimensiones; aquí hay cientos, y «cerca» lo decide el modelo de embeddings, no el sentido común	Una lista de 384 a 1.024 números por cada pedazo de texto
Índice vectorial	El archivador donde se guardan esas coordenadas para buscar «lo que se parece»	Busca parecido, no verdad : puede traer lo parecido y equivocado	Una base de datos (o una tabla) con los embeddings y sus metadatos
Chunk	Cada ficha en la que se recorta un documento para archivarlo	Recortar mal —por tamaño y no por sentido— rompe la información	Pedazos de 200 a 800 tokens con su origen anotado
RAG	Dejarle sobre la mesa solo las páginas relevantes antes de preguntar, en vez de que responda de memoria	Si las páginas que se le dan son las equivocadas, la respuesta es correcta respecto a páginas equivocadas	Recuperar → poner en contexto → responder citando . Módulo 6
Reranking	Un segundo lector que reordena las fichas encontradas y descarta las que no sirven	Cuesta tiempo; en consultas simples no aporta	Un modelo pequeño que puntúa cada chunk contra la pregunta

Piezas que actúan

Término	Símil	Dónde se rompe el símil	Cómo se ve en la práctica
Herramienta (tool)	El empleado puede usar la calculadora, abrir el archivo o mandar un correo —pero solo las que se le dejaron sobre la mesa	Él elige cuándo usarlas y puede elegir mal	Una función con nombre, descripción y un esquema de entradas. Módulo 5
Agente	Un empleado al que se le da una meta y herramientas , y va decidiendo pasos hasta cumplirla	Un empleado real sabe cuándo parar; un agente hay que limitarlo con presupuesto y estados	Un bucle: modelo → decide → usa herramienta → ve el resultado → repite. Módulo 10
Skill	Una carpeta de instrucciones + plantillas que se le entrega para una tarea concreta	—	Texto e instrucciones versionadas, a veces con sus propias herramientas
Harness	El reglamento y el supervisor : qué puede hacer, cuánto puede gastar, qué queda registrado	—	El código que corre al agente: presupuestos, permisos, trazas, estado persistido
Orquestador	El jefe de turno que asigna trabajo y decide qué sigue —con un procedimiento escrito, no improvisando	Si el jefe de turno improvisa, el proyecto fracasa (módulo 10)	Una máquina de estados , no un modelo libre
MCP	El enchufe estándar para que cualquier herramienta se conecte a cualquier modelo	Es un protocolo, no una garantía de calidad de lo que se enchufa	Un servidor que expone herramientas por JSON-RPC; el cliente las descubre. Módulo 0.4
Multimodal / VLM	Un modelo que además ve (imágenes, páginas escaneadas)	Ver no es leer con exactitud: en tablas y cifras falla más que un OCR	Se le pasa una imagen y responde texto. Módulo 7
OCR	Pasar una foto de una página a letras que se pueden buscar	Con sellos, columnas y tablas, lo que sale ya no es la página	Módulo 7: hay cuatro niveles, y el nivel lo decide el documento

Medir y operar

Término	Símil	Dónde se rompe el símil	Cómo se ve en la práctica
VRAM	El tamaño de la mesa de la tarjeta gráfica: si el modelo no cabe, no corre (o corre a paso de tortuga)	—	Gigabytes. El primer número que se mira al comprar. Módulo 1
Ancho de banda	Qué tan rápido se puede leer toda la mesa, una y otra vez	—	GB/s. Decide los tokens por segundo
TTFT	Cuánto espera el empleado antes de decir la primera palabra	—	Milisegundos. Es lo que el usuario <i>siente</i>
Determinista / probabilístico	Una calculadora contra un empleado: la calculadora siempre da lo mismo ; el empleado, casi siempre	—	Regla del curso: el modelo solo donde hay ambigüedad real. Módulo 9
Esquema (JSON Schema)	El formulario con casillas que la respuesta tiene que llenar, ni una casilla más ni menos	—	Se valida cada salida; lo que no cumple se rechaza. Módulo 5
Eval	El examen que se le repite al sistema cada vez que se cambia algo	—	Casos con respuesta esperada y una nota. Módulo 12
Traza	La bitácora : quién pidió qué, qué pasos se dieron, qué fuentes se usaron, cuánto costó	—	Un registro por operación, atribuido a un usuario. Módulo 11
Fine-tuning / LoRA	Mandar al empleado a un curso corto para que hable como la empresa	No le enseña hechos nuevos con fiabilidad; le enseña formas	Módulo 13: casi nunca es la respuesta, y cuando lo es, LoRA
Destilación	Que el empleado experto entrene al aprendiz con ejemplos	—	Un modelo grande genera ejemplos con los que se especializa uno pequeño

Infraestructura

Término	Símil	Dónde se rompe el símil	Cómo se ve en la práctica
Texto plano / Markdown	Escribir en papel en vez de en un formulario de Word: cualquiera lo lee, en cualquier época	Para diagramación fina (un contrato con formato exacto) no alcanza	Archivos .md. Módulo 0.2
Git	El historial de versiones de cada archivo: quién cambió qué, cuándo, y se puede volver atrás	—	Cada cambio queda registrado; los secretos no se guardan ahí
Malla privada	Un cable directo entre los equipos, aunque estén en ciudades distintas, sin abrir la puerta de la calle	—	Nunca se abren puertos al internet. Módulo 3
Proxy inverso	La recepción del edificio: todo entra por ahí y ella decide a qué oficina va	—	Un solo punto de entrada con identidad. Módulo 3
SSO	Una sola credencial para todas las puertas	—	Un usuario, un permiso por grupo, un registro
NAS	El archivador de la oficina: guarda, no piensa	Muchos NAS venden «IA» —es marketing: computan poco	Documentos y copias de seguridad. Módulo 2
Backup 3-2-1	Tres copias, en dos medios, una fuera del edificio	La copia que no se ha probado a restaurar no existe	Módulo 2: la prueba es restaurar

PRÁCTICA · Práctica

1. Elija cinco términos de la tabla y explíquelos en voz alta a alguien que no sea técnico, usando solo el símil. Anote en cuál el símil se rompió antes de tiempo.
2. Tome una noticia reciente sobre IA y subraye cada término de este glosario que aparezca. ¿Cuántos se usan **sin** su sentido medible?
3. Para «alucinación», escriba en una línea qué haría usted para que el empleado del símil dejara de inventar. Esa línea es el módulo 6 completo.

Referencias

1. Vaswani, A. et al. (2017). *Attention Is All You Need*. arXiv:1706.03762 — el artículo del que salen los modelos de lenguaje actuales (transformers).
2. Lewis, P. et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*.

arXiv:2005.11401 — el origen del término RAG.

3. Anthropic (2024). *Building effective agents*. <https://www.anthropic.com/engineering/building-effective-agents> — la distinción entre flujos y agentes que este curso adopta.
4. Model Context Protocol — especificación y documentación: <https://modelcontextprotocol.io/>
5. llama.cpp — formato GGUF y niveles de cuantización: <https://github.com/ggml-org/llama.cpp>

0.2 · Texto plano y Markdown: la base de todo

Objetivos

- Justificar por qué todo el sistema se construye sobre texto versionable
- Escribir un README con la estructura del proyecto

Por qué se empieza por aquí

Todo lo que se construye en este curso —prompts, esquemas, configuraciones, trazas, políticas, documentación de entrega— es **texto**. La decisión más barata y más duradera que se toma al principio es en qué formato vive ese texto. La respuesta es aburrida y correcta: **texto plano**.

Nota Un archivo de texto plano de 1995 se abre hoy con cualquier programa. Un `.doc` de 1995, no siempre. Un proyecto de IA que dure cinco años en una PyME va a cambiar de modelo tres veces y de implementador al menos una. Lo único que sobrevive a todos esos cambios es el texto.

Tres razones concretas, no de gusto:

1. **Se versiona.** Git muestra línea por línea qué cambió, quién y cuándo. Con un binario solo muestra «cambió».
2. **Lo leen las máquinas y las personas.** El mismo `.md` es la documentación para el cliente, el contexto para el modelo y la fuente de la guía impresa.
3. **No depende de nadie.** Sin licencia, sin programa propietario, sin versión que se descontinúe.

Markdown: texto plano con lo mínimo de estructura

Markdown es una convención para marcar títulos, listas, énfasis, tablas y código **usando solo caracteres del teclado**. Se lee sin procesar y se convierte a HTML, PDF o láminas sin esfuerzo.

```
# Título de nivel 1
## Título de nivel 2

Un párrafo con **negrita**, *cursiva* y `código en línea`.

- Lista con viñetas
- Otro ítem
  - Sub-ítem

1. Lista numerada
2. Segundo paso

| Columna | Otra |
|---|---|
| celda | celda |
```

echo "bloque de código con lenguaje declarado"

La versión que se usa en este curso es **CommonMark** con las extensiones de GitHub (tablas, listas de tareas, código con lenguaje). Es la que entiende el MOOC, Git, los editores y casi todos los modelos.

Nota *Markdown* viene de *markup* (lenguaje de marcado) al revés: **marcar hacia abajo**, o sea, marcar lo mínimo. El nombre es una broma de su autor, John Gruber (2004).

Front-matter: metadatos al principio del archivo

Muchas herramientas aceptan un bloque de **YAML** entre dos líneas de `---` al inicio del archivo. Es donde va lo que describe al documento sin ser el documento:

```
---
titulo: "Asistente documental – ferretería El Tornillo"
version: 0.1
cliente: ferreteria-el-tornillo
verificado: 2026-09
---

# Asistente documental
...
```

Ese bloque es lo que va a leer el importador del MOOC, el generador de la guía impresa y el

propio agente cuando necesite saber de qué trata un archivo. **Un mismo texto, varios lectores.**

Por qué el texto plano también es la base para el modelo

Un modelo lee tokens (lección siguiente). Todo lo que no es texto —una imagen, un PDF escaneado, una hoja de cálculo con colores— tiene que **convertirse** a texto antes de que el modelo lo vea, y en esa conversión se pierde información (módulo 7 muestra cuánta). Cuando la fuente ya es texto plano bien estructurado, no hay conversión y no hay pérdida.

De ahí una regla práctica del curso:

ATENCIÓN · Ojo

Lo que el sistema produce, lo produce en texto plano estructurado. Actas, políticas, extracciones, informes. Si el cliente quiere Word o PDF, se genera **desde** el texto, nunca al revés.

Cuándo NO usar esto Texto plano y Markdown: la base de todo

- **Documentos con diagramación exacta y valor legal** (un contrato con márgenes, firmas y foliación): se producen en PDF desde una plantilla; el Markdown es el borrador, no la entrega.
- **Datos tabulares grandes:** una tabla Markdown de 2.000 filas no sirve para nada. Eso es CSV o una base de datos; Markdown solo para la tabla resumen.
- **Formularios que llena gente no técnica:** nadie va a escribir | celda | a mano. Se les da una interfaz, y la interfaz guarda texto plano por debajo.
- **Diagramas complejos:** Mermaid (texto → diagrama) sirve hasta cierto tamaño; un plano de red de 40 nodos se dibuja en otra herramienta y se exporta a SVG.

PROCEDIMIENTO · Laboratorio 0.2 · el README del proyecto

El proyecto integrador del curso empieza aquí: una carpeta con un solo archivo. Cada módulo va a añadirle una capa.

```
mkdir -p ~/labs/el-tornillo && cd ~/labs/el-tornillo
git init -q
cat > README.md <<'EOF'
---
proyecto: asistente-documental
cliente: Ferretería El Tornillo (Bucaramanga)
```

```
usuarios: [gerente, contadora, vendedor]
```

```
verificado: 2026-09
```

```
---
```

```
# Asistente documental – Ferretería El Tornillo
```

```
## Qué hace (versión 0)
```

```
Todavía nada. Este archivo es la primera capa: la descripción del sistema.
```

```
## Qué va a hacer
```

1. Ingerir documentos escaneados (facturas, remisiones, contratos).
2. Extraer campos con esquema y validarlos.
3. Responder consultas citando la fuente.
4. Dejar traza auditable de cada operación.
5. Servir a tres empleados con permisos distintos.

```
## Perfiles de hardware en los que se prueba
```

- A · portátil sin GPU dedicada
- B · equipo con ~816 GB de VRAM
- C · servidor con 24 GB o más

```
EOF
```

```
git add README.md && git commit -qm "Capa 0: descripción del sistema" && git log --oneline
```

Lo que debe ver: una línea con el hash del commit y el mensaje Capa 0: descripción del sistema.

Entregable: el README.md versionado. De aquí en adelante, cada laboratorio termina con un commit cuyo mensaje empieza por «Capa N».

Qué se degrada en cada perfil

A — Igual. Es texto

B — Igual

C — Igual

Es el único laboratorio del curso donde los tres perfiles son idénticos. Conviene fijarse en eso: **la parte que más dura del sistema es la que menos hardware necesita.**

PRÁCTICA · Práctica

1. Convierta a Markdown la última acta o informe que haya escrito en Word. Anote qué se perdió (probablemente nada importante) y qué se ganó.
2. Abra el README.md con tres programas distintos (un editor de código, el bloc de notas, el

navegador vía GitHub). Debe leerse en los tres.

3. Escriba el front-matter de un documento del cliente: título, versión, quién lo aprobó, fecha de verificación.

Referencias

1. CommonMark — especificación: <https://commonmark.org/> · GitHub Flavored Markdown: <https://github.github.com/gfm/>
2. Gruber, J. (2004). *Markdown*. <https://daringfireball.net/projects/markdown/> — el documento original.
3. Hunt, A. y Thomas, D. (1999). *The Pragmatic Programmer*. Addison-Wesley — capítulo «The Power of Plain Text».
4. YAML — especificación 1.2: <https://yaml.org/spec/1.2.2/>

0.3 · Modelo, pesos, cuantización, contexto y tokens

Objetivos

- Definir cada término de forma medible
- Contar tokens de documentos reales y estimar el costo de contexto

Cinco palabras, cinco números

Cada término de esta lección se define por **un número que se puede medir en la terminal**. Si alguien los usa sin número, está hablando de otra cosa.

Término	El número	Cómo se mide
Modelo (tamaño)	parámetros: 1B, 8B, 70B...	está en el nombre del archivo
Pesos en disco	gigabytes	<code>ls -lh</code>
Cuantización	bits por parámetro: 16, 8, 4...	el sufijo: F16, Q8_0, Q4_K_M
Contexto	tokens que caben en la ventana	lo declara el runtime al cargar
Tokens	cuántos pedazos tiene un texto	un programa tokenizador

Modelo y pesos

Un **modelo de lenguaje** es una función enorme que recibe una secuencia de tokens y devuelve, para el siguiente token, una probabilidad por cada token posible del vocabulario. Se muestrea

uno, se añade a la secuencia, y se repite. Eso es todo lo que hace. Todo lo demás —«razonar», «resumir», «extraer»— es ese bucle aplicado muchas veces.

Los **pesos** (o parámetros) son los números que definen esa función. Un modelo de 8B tiene ocho mil millones. En **precisión completa** (16 bits por parámetro) ocupan:

8 000 000 000 parámetros × 2 bytes = 16 GB

Ese cálculo es la primera herramienta de decisión del curso: **parámetros × bytes por parámetro = memoria que hace falta solo para cargar el modelo**, sin contar el contexto.

Cuantización

Cuantizar es guardar cada peso con **menos bits**. Los formatos más comunes en runtimes locales [VOLÁTIL – verificado: 2026-09]:

Sufijo	Bits/parámetro	8B ocupa	Pérdida típica	Cuándo
F16 / BF16	16	~16 GB	ninguna (referencia)	evaluación, entrenamiento
Q8_0	8	~8,5 GB	casi imperceptible	cuando cabe
Q6_K	~6,5	~6,6 GB	muy baja	buen punto medio
Q4_K_M	~4,8	~4,9 GB	notable en tareas finas	perfil B por defecto
Q3 / Q2	3–2	~3,5–3 GB	alta, a veces inutilizable	solo si no hay otra

ATENCIÓN · Ojo

«Notable en tareas finas» no es una opinión: **se mide** con el mismo set de preguntas en varias cuantizaciones (laboratorio 4.2). Un modelo que extrae cifras de facturas puede pasar de 97 % de exactitud en Q8 a 91 % en Q4. Ese 6 % son facturas mal leídas.

Criterio de selección (no un nombre de modelo): elija la cuantización más alta que **quepa** en la VRAM dejando espacio para el contexto; baje un escalón solo si la medición dice que la pérdida es aceptable para la tarea.

Contexto: la ventana

La **ventana de contexto** es la cantidad máxima de tokens que el modelo puede tener «a la vista» a la vez: instrucciones + documentos + conversación + su propia respuesta. Todo compite por ese espacio.

Y tiene un costo que casi nadie cuenta al comprar hardware: **la caché de atención (KV cache)** crece con el contexto y vive en la misma memoria que los pesos. Como orden de magnitud para un modelo 8B [VOLÁTIL – verificado: 2026-09]:

```
contexto de 8 000 tokens → ~1 GB adicional
contexto de 32 000 tokens → ~4 GB adicionales
```

Por eso «cabe en 8 GB» puede ser cierto para el modelo solo y falso para el modelo con un PDF de 40 páginas dentro. El módulo 1 lo convierte en fórmula.

Tokens

Un **token** es la unidad en que el modelo lee: no palabras, no letras, sino fragmentos decididos por un algoritmo de compresión estadística (BPE o similar) durante el entrenamiento. Consecuencias prácticas:

- El español gasta **más tokens por palabra** que el inglés, porque los tokenizadores se entrenan con mayoría de inglés. Regla de bolsillo: 1.000 palabras en español $\approx$ 1.400–1.800 tokens.
- Los números, los códigos de factura y las siglas se parten en muchos tokens: FV-2026-000123 puede ser 8 tokens.
- Cada modelo tiene **su** tokenizador. Contar con el de otro da cifras equivocadas.

PROCEDIMIENTO · Laboratorio 0.3 · contar tokens de documentos reales

Se usa la biblioteca tokenizers con el archivo tokenizer.json de **cualquier** modelo: así el laboratorio no depende de un modelo concreto. (Si ya tiene un runtime local, la mayoría expone su propio conteo; el principio es el mismo.)

```
cd ~/labs/el-tornillo && mkdir -p labs/00-03 && cd labs/00-03
python3 -m venv .venv && source .venv/bin/activate
pip install -q tokenizers
# tokenizer.json: descárguelo del repositorio del modelo que vaya a usar (Hugging Face) y
# póngalo aquí. Es un archivo de texto; no hacen falta los pesos.

# tokens.py - cuenta tokens de varios archivos con un tokenizer.json cualquiera
import sys, pathlib
from tokenizers import Tokenizer

tok = Tokenizer.from_file("tokenizer.json")
for ruta in sys.argv[1:]:
    texto = pathlib.Path(ruta).read_text(encoding="utf-8")
    n_tok = len(tok.encode(texto).ids)
    n_pal = len(texto.split())
    print(f"{ruta:40s} {n_pal:7d} palabras {n_tok:8d} tokens   ratio {n_tok/max(n_pal,1):.2f}")

python tokens.py ../../README.md docs/politica-datos.md docs/contrato-proveedor.md
```

Lo que debe ver: tres líneas con el ratio tokens/palabra. En español bien redactado sale entre **1,4 y 1,8**. Si sale 2,5 o más, el texto tiene muchos códigos, números o mayúsculas raras —y eso es lo que va a costar en contexto.

Entregable: labs/00-03/tokens.py y una tabla con los tokens de los tres documentos más largos del cliente. Commit: «Capa 0.3: presupuesto de tokens».

Qué se degrada en cada perfil

A — Igual: tokenizar es CPU

B — Igual

C — Igual

Cuándo NO usar esto Modelo, pesos, cuantización, contexto y tokens

- **No cuente tokens con el tokenizador de otro modelo** para presupuestar contexto: las cifras pueden diferir un 30 %.
- **No cuantice a Q2/Q3 para «que quepa»** en un equipo que no da: mida primero; casi siempre es mejor un modelo más pequeño en Q8 que uno grande en Q2.
- **No compre por parámetros.** Un 70B en Q2 puede rendir peor que un 8B en Q8 y costar cuatro veces más de memoria.

PRÁCTICA · Práctica

1. Calcule cuánta memoria ocupa un modelo de 14B en F16, Q8 y Q4 (sin contexto).
2. Un cliente tiene facturas de 3 páginas (~1.200 palabras). ¿Cuántos tokens aproximados gasta una sola factura? ¿Cuántas caben en una ventana de 8k dejando 1.500 para instrucciones y respuesta?
3. Explique en dos líneas por qué un runtime puede decir «modelo cargado» y aun así fallar al procesar un documento largo.

Referencias

1. llama.cpp — formato GGUF y tabla de cuantizaciones: <https://github.com/ggml-org/llama.cpp> (README y docs/). [VOLÁTIL – verificado: 2026-09]
2. Hugging Face — biblioteca tokenizers: <https://huggingface.co/docs/tokenizers>

3. Sennrich, R. et al. (2016). *Neural Machine Translation of Rare Words with Subword Units*. arXiv:1508.07909 — el BPE que usan los tokenizadores modernos.
4. Detrmers, T. et al. (2023). *QLoRA: Efficient Finetuning of Quantized LLMs*. arXiv:2305.14314 — la evidencia de que 4 bits conserva la mayor parte de la calidad en muchas tareas.

0.4 · Embeddings, herramienta, agente, skill, harness, MCP y multimodal

Objetivos

- Distinguir cada pieza por lo que hace
- Ejecutar un ejemplo mínimo de cada una

Siete piezas, cada una con un ejemplo que corre

Esta lección cierra el vocabulario con las piezas que **actúan**. Cada una se define por lo que hace y se demuestra con el ejemplo más pequeño que corre. Nada de esto necesita GPU.

1 · Embedding

Un **embedding** es un vector de números que representa el significado de un texto, de modo que textos parecidos tienen vectores cercanos. Lo produce un **modelo de embeddings**, distinto del modelo de lenguaje y mucho más pequeño.

```
# embed.py – parecido entre frases con un modelo de embeddings local
# [VOLÁTIL – verificado: 2026-09] el nombre del modelo cambia; el criterio no:
# multilingüe con buen español, ≤ 600 MB, licencia permisiva. Dos alternativas por tamaño
# se listan en el laboratorio 6.2.
from sentence_transformers import SentenceTransformer
m = SentenceTransformer("<modelo-de-embeddings-multilingue>")
frases = ["factura del proveedor de tornillería", "remisión de tornillos", "acta de reunión del
comité"]
v = m.encode(frases, normalize_embeddings=True)
print((v @ v.T).round(2)) # matriz de similitudes coseno
```

Lo que debe ver: las dos primeras frases con similitud alta entre sí ($\approx 0,7-0,9$) y baja con la tercera. Eso —y no la magia— es lo que hace funcionar un RAG.

ATENCIÓN · Ojo

Un embedding mide parecido de forma, no verdad. «El pago fue aprobado» y «el pago no fue aprobado» quedan **cerca**. El módulo 6 explica cómo se compensa.

2 · Herramienta (tool)

Una **herramienta** es una función con tres cosas: un nombre, una descripción en lenguaje natural y un **esquema** de entradas. El modelo no la ejecuta: **pide** que se ejecute, con argumentos que cumplen el esquema, y recibe el resultado como texto.

```
# tool.py – una herramienta es una función + su contrato
HERRAMIENTAS = [{
    "name": "buscar_factura",
    "description": "Busca una factura por número exacto. Devuelve proveedor, fecha y total.",
    "parameters": {"type": "object",
        "properties": {"numero": {"type": "string", "pattern": "^FV-\\d{4}-\\d{6}$"}},
        "required": ["numero"]}
}]

def buscar_factura(numero: str) -> dict:
    return {"numero": numero, "proveedor": "Tornillería Santander", "fecha": "2026-08-14", "total": 1850000}
```

Regla de diseño (módulo 5): descripción exacta, entradas pequeñas y validadas, salida en JSON. Una herramienta que acepta «texto libre» es una herramienta que va a fallar.

3 · Agente

Un **agente** es un bucle: el modelo ve el estado, decide (responder o usar una herramienta), la herramienta corre, el resultado vuelve al contexto, y se repite hasta cumplir la meta **o agotar el presupuesto**. Sin presupuesto no es un agente: es un riesgo.

```
# agente.py – el bucle mínimo, con presupuesto duro
def agente(modelo, herramientas, meta, max_pasos=6):
    contexto = [{"rol": "sistema", "texto": "Cumpla la meta usando herramientas. Responda FIN cuando termine."},
        {"rol": "usuario", "texto": meta}]
    for paso in range(max_pasos):
        decision = modelo(contexto, herramientas)          # → {"tipo": "texto"|"tool", ...}
        if decision["tipo"] == "texto":
            return decision["texto"]
        resultado = ejecutar(decision["tool"], decision["args"])
        contexto.append({"rol": "tool", "texto": str(resultado)})
    return "PRESUPUESTO AGOTADO – se requiere revisión humana"
```

El módulo 10 convierte este bucle en una **máquina de estados** con estado persistido, que es la

versión que aguanta producción.

4 · Skill

Una **skill** es un paquete versionado de instrucciones (y, a veces, herramientas y plantillas) para una tarea concreta: «extraer campos de una factura de proveedor colombiano», «redactar un acta con este formato». No es código nuevo: es **contexto empaquetado** que se carga cuando hace falta y se descarga cuando no —que es la forma de no llenar la ventana con todo a la vez.

```
skills/extraer-factura/|—
  SKILL.md           ← qué hace, cuándo se usa, pasos, ejemplos|—
  esquema.json       ← el JSON Schema de la salida|—
  ejemplos/          ← 3 facturas anotadas
```

5 · Harness

El **harness** es el programa que corre al agente y le pone las reglas: qué herramientas puede ver, cuánto puede gastar (pasos, tokens, tiempo, dinero), qué queda en la traza, dónde se guarda el estado, y **en qué punto se detiene a pedir aprobación humana**. Es la parte del sistema que el cliente compra sin saberlo: lo que hace que el agente sea auditable.

6 · MCP

Model Context Protocol es un protocolo abierto para que una herramienta se exponga una vez y la use cualquier cliente compatible. Un **servidor MCP** publica herramientas (y recursos) por JSON-RPC; el cliente las descubre y las llama. Es el enchufe estándar del glosario.

```
# Un servidor MCP mínimo que expone buscar_factura. [VOLÁTIL – verificado: 2026-09]
pip install -q mcp
```

```
# servidor_mcp.py
from mcp.server.fastmcp import FastMCP
mcp = FastMCP("el-tornillo")

@mcp.tool()
def buscar_factura(numero: str) -> dict:
    """Busca una factura por número exacto (FV-AAAA-NNNNNN)."""
    return {"numero": numero, "proveedor": "Tornillería Santander", "total": 1850000}

if __name__ == "__main__":
    mcp.run()
```

Lo que debe ver: al conectar cualquier cliente MCP, aparece `buscar_factura` con su descripción y su esquema, sin haber escrito nada específico para ese cliente.

7 · Multimodal y modelos de visión (VLM)

Un modelo **multimodal** acepta más de un tipo de entrada —texto e imagen, y en algunos casos audio—. Un **VLM** (modelo de visión y lenguaje) recibe una imagen y responde texto: describe, transcribe, localiza. Sirve para páginas escaneadas con layouts complejos, **pero no reemplaza al OCR** en cifras y tablas: el módulo 7 mide dónde gana cada uno.

Cómo encajan las siete piezas en el proyecto

- Los documentos se **embeben** y se indexan (módulo 6).
- El asistente es un **agente** con **herramientas** pequeñas (módulos 5 y 10).
- Cada tarea del cliente es una **skill** versionada.
- El **harness** pone presupuestos, permisos y trazas (módulos 10–12).
- Las herramientas se exponen por **MCP** para que cualquier interfaz las use.
- Las páginas escaneadas pasan por el ruteo OCR/**VLM** (módulo 7).

Cuándo NO usar esto Embeddings, herramienta, agente, skill, harness, MCP y multimodal

- Un agente para una tarea que un flujo determinista resuelve (mover un archivo, renombrar, sumar una columna). Módulo 9: el modelo solo donde hay ambigüedad.
- Embeddings para buscar un número exacto (una cédula, un número de factura). Eso es una búsqueda exacta en base de datos; los embeddings traerán «parecidos».
- MCP para una sola herramienta que usa un solo programa. Es un enchufe: vale la pena cuando hay más de un aparato.

PRÁCTICA · Práctica

1. Escriba el contrato (nombre, descripción, esquema) de la herramienta `listar_facturas_de` que recibe un proveedor y un rango de fechas. Rechace al menos dos entradas mal formadas.
2. En el bucle del agente, ¿qué pasa si `max_pasos` es 100? Describa dos daños concretos.
3. Dé un ejemplo del negocio del cliente donde una skill sea mejor que «meter todas las instrucciones en el prompt».

Referencias

1. Anthropic (2024). *Building effective agents*. <https://www.anthropic.com/engineering/building-effective-agents>
2. Model Context Protocol — especificación: <https://modelcontextprotocol.io/> · SDK de Python: <https://github.com/modelcontextprotocol/python-sdk> [VOLÁTIL – verificado: 2026-09]
3. Reimers, N. y Gurevych, I. (2019). *Sentence-BERT*. arXiv:1908.10084 — base de los modelos de embeddings de frases. Biblioteca: <https://www.sbert.net/>
4. JSON Schema — especificación: <https://json-schema.org/>
5. Yao, S. et al. (2022). *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv:2210.03629 — el patrón «pensar → actuar → observar» del bucle de agente.