IA local para PyMEs · Módulo 10

La organización de agentes

Por qué «empresa con CEO» fracasa y cuál es la versión válida. Orquestador como máquina de estados, contratos JSON, humano en el punto irreversible.

Qué se construye en este módulo

Por qué la metáfora «empresa con CEO» hace fracasar proyectos

Hacer la autopsia de un diseño multiagente fallido

Las cuatro razones legítimas para dividir en roles

Justificar cada rol por permisos, contexto, modelo o paralelismo

El orquestador como máquina de estados

Escribir la máquina de estados del asistente

Qué se construye en este módulo

Contratos JSON entre roles y el humano en el punto irreversible

Definir contratos entre roles

10.1 · Por qué la metáfora «empresa con CEO» hace fracasar proyectos

Por qué la metáfora «empresa con CEO» hace fracasar proyectos

«Un agente CEO que delega en un agente de ventas, uno de contabilidad y uno de compras, y ellos se coordinan entre sí.» Suena bien en una diapositiva. En producción fracasa casi siempre, por razones que no son de moda sino de ingeniería:

En una tabla

Lo que la metáfora supone	Lo que pasa
Los roles «entienden» su cargo	entienden un prompt; con contexto insuficiente se salen
Se coordinan hablando	el texto libre entre agentes pierde información y multipli
El CEO decide bien	el CEO es un modelo sin estado, sin memoria de lo que ya
Más agentes = más capacidad	más agentes = más prefill, más latencia, más puntos de
Se supervisan entre sí	dos modelos que se equivocan igual no se corrigen
Es «como una empresa»	una empresa tiene procedimientos escritos, estado en s humanos que paran cosas

Cuándo NO usar esto

Por qué la metáfora «empresa con CEO» hace fracasar proyectos

- No use esta lección para decir «los agentes no sirven». Sirven cuando hay razones legítimas para dividir (lección siguiente) y un orquestador que no improvisa.

10.2 · Las cuatro razones legítimas para dividir en roles

Las cuatro razones legítimas para dividir en roles

Si un rol no se justifica con **al menos una** de estas, no existe: es prompt.

En una tabla

Razón	Qué resuelve	Ejemplo en la ferretería
1 · Permisos	un rol puede ver o hacer cosas que otro no debe	el rol que lee nómina no es el mismo que atiende al vendedor; el que **escribe** en la base es distinto del que **lee**
2 · Contexto	dos tareas necesitan contextos que no caben juntos o se estorban	extraer facturas (esquema + ejemplos de facturas) y redactar actas (plantilla + transcripción) no comparten ventana
3 · Modelo	tareas distintas rinden mejor con modelos distintos	clasificar el tipo de documento: modelo pequeño y rápido; extraer campos: modelo mediano; VLM solo para fotos
4 · Paralelismo	volumen: muchas unidades independientes a la vez	400 facturas de fin de mes: N auxiliares idénticos, cada uno con una factura

Cuándo NO usar esto

Las cuatro razones legítimas para dividir en roles

- No cree un rol «para que quede ordenado». Orden es un directorio, no un agente.
- No dé al auditor permisos de escritura: deja de ser auditor.

Qué se degrada en cada perfil

A · sin GPU

Razón 3 manda: casi todo con el modelo pequeño; razón 4 no aplica (no hay paralelismo)

B · 8–16 GB VRAM

Dos modelos cargados a la vez si caben; paralelismo bajo

C · 24 GB+

Todos los roles con su modelo; paralelismo real

10.3 · El orquestador como máquina de estados

El orquestador como máquina de estados

Un **orquestador** es la pieza que decide qué paso sigue. Si esa pieza es un modelo con un prompt («coordina a los agentes»), el sistema es tan fiable como la peor respuesta de ese modelo. Si es una **máquina de estados** escrita en YAML y ejecutada por código, es tan fiable como el código.

En una tabla

Perfil	Qué pasa
A	Igual; cada paso con modelo tarda más, y el presupuesto de segundos se ajusta
B / C	Igual

Cuándo NO usar esto

El orquestador como máquina de estados

- No ponga un modelo a «decidir el siguiente estado» salvo dentro de un estado, con salida enum y esquema. La transición la hace el código.
- No omita el estado `fallida`: un sistema que no puede fallar limpiamente falla sucio.

10.4 · Contratos JSON entre roles y el humano en el punto irreversible

Contratos JSON entre roles y el humano en el punto irreversible

Lo que un rol le pasa a otro es un **contrato JSON** con esquema, no un párrafo. Tres razones: se valida, se registra, y cuando cambia el modelo de un rol el otro no se entera.

En una tabla

Acción	¿Reversible?	Quién
crear borrador de acta	sí	sistema
registrar factura en contabilidad	**no**	contadora aprueba
enviar correo al proveedor	**no**	gerente aprueba
mover PDF a `procesados/`	sí	sistema
borrar datos de un titular (2.3)	**no**	gerente aprueba +

El comando, copiable

```
{ "$id": "contrato/→extractorvalidador",
  "type": "object", "additionalProperties": false,
  "required": ["tarea_id", "documento", "campos", "confianza", "origen"],
  "properties": {
    "tarea_id":  {"type": "string"},
    "documento": {"type": "string", "description": "id del original, nunca el
     contenido"},
    "campos":    {"$ref": "factura.schema.json"},
    "confianza": {"type": "object"},
    "origen":    {"type": "object", "properties": {"rol": {"type":"string"}, "modelo"
     : {"type":"string"}, "version": {"type":"string"}}}
  }}
```

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

Cuándo NO usar esto

Contratos JSON entre roles y el humano en el punto irreversible

- No pida aprobación para todo: si el 90 % de las facturas pasan por la contadora, el sistema no ahorra nada. El umbral se ajusta con 7.4.
- No deje que el modelo «proponga» la decisión en la cola con texto persuasivo: muestra evidencia, no argumentos.

Entregables del módulo

- `labs/10-01/autopsia.md`: tome un flujo multiagente publicado (o el propio) y escriba su autopsia con la tabla de arriba.
- la tabla justificada.
- `maquina.yaml`, `ejecutar.py`, y el registro de transiciones de las 20.
- `contratos/*.schema.json` para los 5 pares de roles, y la cola funcionando con los 3 usuarios.

Por qué «empresa con CEO» fracasa y cuál es la versión válida.