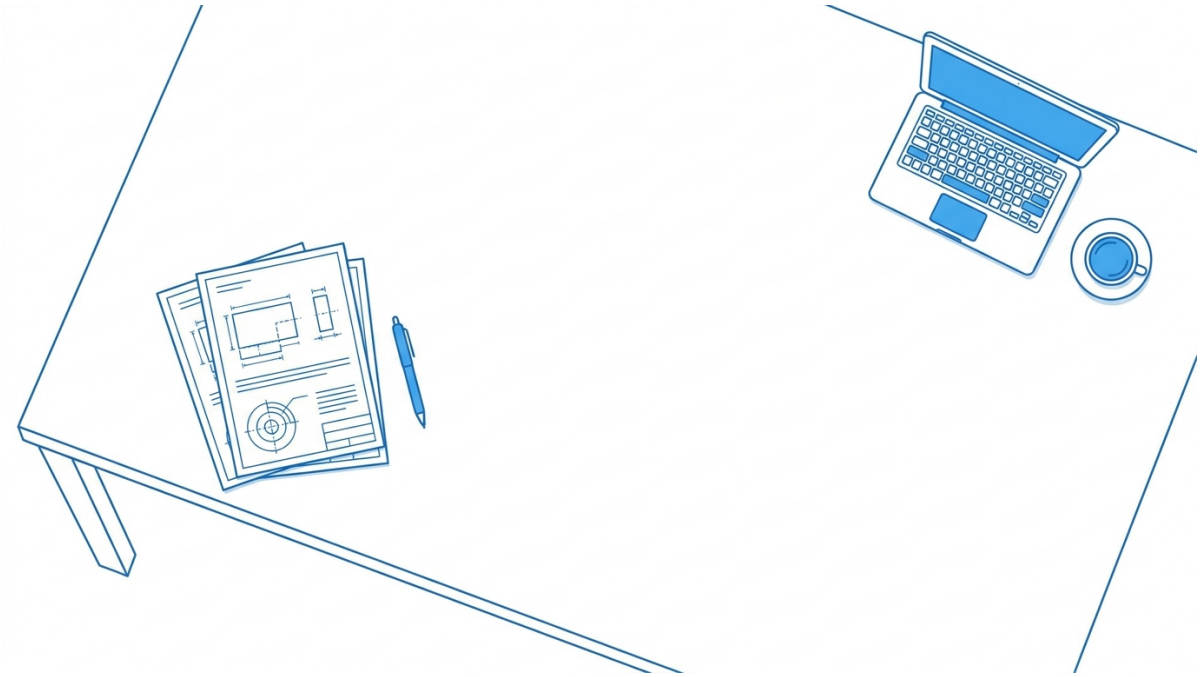
> **LO ESENCIAL · Módulo 5 · Ingeniería de contexto**
>
> La ventana es un recurso escaso. Salidas con esquema, herramientas pequeñas y poda en agentes largos.

## 5.1 · De prompts a contexto: la ventana como recurso escaso



Ingeniería de contexto

> Objetivos
>
> - Presupuestar la ventana por secciones
> - Explicar por qué la ingeniería de prompts se quedó corta

### Por qué «ingeniería de prompts» se quedó corta

Durante dos años el oficio fue redactar bien la instrucción. Sigue importando, pero dejó de ser el problema principal. El problema principal es **qué entra en la ventana, en qué orden, y qué se saca**: instrucciones, documentos recuperados, historial, resultados de herramientas, esquemas.

Eso es **ingeniería de contexto**: tratar la ventana como un recurso escaso y presupuestado.

Tres hechos la vuelven obligatoria:

1. **Cuesta.** Cada token de entrada se paga en prefill, en cada paso (módulo 1.2).
2. **Se degrada.** Los modelos atienden peor a lo que queda en la mitad de contextos largos («perdido en el medio»). Más contexto no es más calidad.
3. **Compite.** Lo que ocupan las instrucciones no lo ocupan los documentos, y viceversa.

## El presupuesto de la ventana

Como un presupuesto de caja: cada partida tiene un tope y una prioridad.

| Partida | Qué es | Tope típico (ventana 8k) | Prioridad |
|---|---|---|---|
| Instrucciones del sistema | quién es, qué puede, formato de salida | 600–1 200 | fija |
| Esquema de salida | el JSON Schema, si aplica | 200–500 | fija |
| Contexto recuperado (RAG) | chunks relevantes con su cita | 3 000–4 500 | variable |
| Historial de la conversación | turnos anteriores, podados | 500–1 500 | variable |
| Resultados de herramientas | último resultado, resumido | 300–800 | variable |
| Reserva para la respuesta | lo que el modelo va a escribir | 800–1 500 | fija |

La suma no puede pasar de la ventana **menos** un margen del 10 %. Cuando no cabe, se poda por prioridad, nunca «lo que salga».

## Orden dentro de la ventana

El orden afecta dos cosas: la caché de prefijos (módulo 4.1) y la atención del modelo.

```
[instrucciones fijas]  ← idéntico en todas las llamadas: se cachea
[esquema fijo]         ← ídem
[documentos]           ← cambia por consulta
[historial podado]
[pregunta actual]      ← lo más importante, al final, cerca de la respuesta
```

> PROCEDIMIENTO · **Laboratorio 5.1 · el presupuesto del asistente**
>
> `labs/05-01/presupuesto.md`: para cada una de las tres tareas del asistente (extraer factura, responder consulta con cita, redactar acta), una tabla con las seis partidas, el tope en tokens, y la suma comparada con la ventana del modelo elegido. Use `tokens.py` (0.3) para medir las instrucciones reales.
>
> **Entregable:** el presupuesto de las tres tareas. Commit: «Capa 5.1: presupuesto de contex-

> to».

### Qué se degrada en cada perfil

**A** — RAG con 2–3 chunks; sin historial largo

**B** — RAG con 5–8 chunks; historial podado

**C** — holgura; aun así se presupuesta, porque el prefill se paga

**Cuándo NO usar esto** **De prompts a contexto: la ventana como recurso escaso**

- No «arregle» un mal RAG metiendo más chunks: el módulo 6 dice cómo arreglar la recuperación.
- No use la ventana de 128k como excusa para no presupuestar: el prefill de 100k tokens en el perfil B tarda minutos.

**PRÁCTICA** · **Práctica**

1. El asistente tiene 1 800 tokens de instrucciones. ¿Cuántos chunks de 400 caben en 8k con reserva de 1 000 para la respuesta y 600 de historial?
2. Reordene un prompt que tiene la pregunta al principio. ¿Qué gana?

### Referencias

1. Liu, N. F. et al. (2023). *Lost in the Middle: How Language Models Use Long Contexts*. arXiv:2307.03172.

2. Anthropic — serie de ingeniería sobre agentes y contexto: https://www.anthropic.com/engineering `[VOLÁTIL — verificado: 2026-09]`

3. Módulo 1.2 y 4.1 de este curso — el costo del prefill y la caché de prefijos.

## 5.2 · Salidas estructuradas y validación por esquema

> Objetivos
>
> - Definir un JSON Schema y validar cada salida
> - Rechazar texto libre donde cabe un esquema

## Texto libre donde cabe un esquema es un error de diseño

Si la respuesta tiene estructura —campos de una factura, una decisión sí/no con motivo, una lista de acciones—, se pide **con esquema** y se **valida**. Lo que no cumple el esquema se rechaza y se reintenta o se encola para un humano. Nunca se «parsea» texto libre con expresiones regulares esperando lo mejor.

Dos capas, siempre las dos:

1. **Salida estructurada**: el runtime restringe la generación a un JSON válido según el esquema (la mayoría lo soporta hoy con `response_format` o gramáticas `[VOLÁTIL — verificado: 2026-09]`).
2. **Validación posterior** con una biblioteca de JSON Schema: la restricción del runtime garantiza sintaxis, no semántica (un total negativo es JSON válido).

## El esquema de la factura

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "FacturaProveedor",
  "type": "object",
  "additionalProperties": false,
  "required": ["numero", "fecha", "proveedor", "nit", "subtotal", "iva", "total", "confianza"],
  "properties": {
    "numero":    {"type": "string", "pattern": "^[A-Z]{1,4}-?\\d{1,10}$"},
    "fecha":     {"type": "string", "format": "date"},
    "proveedor": {"type": "string", "minLength": 3, "maxLength": 120},
    "nit":       {"type": "string", "pattern": "^\\d{6,10}(-\\d)?$"},
    "subtotal":  {"type": "integer", "minimum": 0},
    "iva":       {"type": "integer", "minimum": 0},
    "total":     {"type": "integer", "minimum": 0},
    "confianza": {"type": "object", "additionalProperties": {"type": "number", "minimum": 0, "
    maximum": 1}}
  }
}
```

Tres decisiones que valen más que el modelo:

- `additionalProperties: false`: el modelo no puede inventar campos.

- Montos como **enteros en pesos**, no decimales ni strings: se suman sin ambigüedad.

- `confianza` por campo: el modelo declara cuánto confía; lo que está por debajo de 0,8 va a revisión humana (módulo 10.4).

PROCEDIMIENTO · **Laboratorio 5.2 · extraer y validar**

```
cd ~/labs/el-tornillo && mkdir -p labs/05-02 && cd labs/05-02 && pip install -q jsonschema

# extraer.py — pide salida con esquema y la valida; lo que no cumple, se reintenta una vez
     y se encola
import json, requests, jsonschema, sys
ESQ = json.load(open("factura.schema.json"))
URL = "http://localhost:8080/v1/chat/completions"
def extraer(texto, intentos=2):
    for i in range(intentos):
        r = requests.post(URL, json={"model":"local","temperature":0,
             "response_format":{"type":"json_schema","json_schema":{"name":"
    FacturaProveedor","schema":ESQ}},
             "messages":[{"role":"system","content":"Extraiga los campos de la factura.
    Montos en pesos, enteros. Si un campo no está, confianza 0."},
                         {"role":"user","content":texto}]}).json()
        salida = json.loads(r["choices"][0]["message"]["content"])
        try:
            jsonschema.validate(salida, ESQ)
            if salida["subtotal"] + salida["iva"] != salida["total"]:
                raise ValueError("subtotal + iva ≠ total")
            return salida
        except (jsonschema.ValidationError, ValueError) as e:
            print(f"intento {i+1} rechazado: {e}", file=sys.stderr)
    return {"_revision_humana": True, "_motivo": "no cumple esquema tras 2 intentos"}
print(json.dumps(extraer(open(sys.argv[1]).read()), ensure_ascii=False, indent=2))
```

Corra sobre las 20 facturas del set. Lo que debe ver: JSON válidos, y **algunos rechazos** —eso es el sistema funcionando, no fallando. Cuente cuántos, y por qué.
**Entregable:** `factura.schema.json`, `extraer.py`, y `labs/05-02/rechazos.md` con los motivos. Commit: «Capa 5.2: salida con esquema».

### Qué se degrada en cada perfil

**A** — Igual, más lento. La validación es CPU

**B** — Igual

**Cuándo NO usar esto** **Salidas estructuradas y validación por esquema**

- No ponga esquema a una tarea de redacción (un acta, un resumen): ahí la salida es texto, y se valida por longitud y por presencia de secciones, no por JSON.
- No confíe en la restricción del runtime para reglas de negocio (`subtotal + iva = total`): eso se valida en código.

**PRÁCTICA** · **Práctica**

1. Añada al esquema una regla que rechace facturas con fecha futura.
2. ¿Qué hace el sistema con una factura sin NIT visible? Diseñe la respuesta sin inventar.

### Referencias

1. JSON Schema 2020-12 — https://json-schema.org/specification
2. `jsonschema` para Python — https://python-jsonschema.readthedocs.io/
3. llama.cpp — gramáticas GBNF y `response_format`: https://github.com/ggml-org/llama.cpp `[VOLÁTIL — verificado: 2026-09]`

## 5.3 · Diseño de herramientas y poda de contexto

> Objetivos
>
> - Escribir herramientas pequeñas con contratos exactos
> - Podar contexto en un agente largo sin perder estado

### Herramientas pequeñas, descripciones exactas

Un modelo elige herramientas leyendo sus **descripciones**. Una descripción vaga produce llamadas equivocadas; una herramienta que hace tres cosas produce argumentos mezclados. Reglas:

| Regla | Mal | Bien |
|---|---|---|
| Una herramienta, una acción | `gestionar_factura(accion, ...)` | `buscar_factura`, `marcar_pagada`, `listar_pendientes` |
| Entradas mínimas y tipadas | `consulta: string` | `numero: string (patrón)`, `desde: date`, `hasta: date` |
| Descripción con cuándo **no** usarla | «Busca facturas» | «Busca UNA factura por número exacto. Para rangos de fechas use `listar_facturas`» |
| Salida en JSON, corta | vuelca 200 filas | devuelve 20 con `total_disponible` y un cursor |
| Errores como datos | lanza excepción | `{"error": "no_encontrada", "sugerencia": "verifique el prefijo"}` |
| Solo lectura por defecto | cualquier herramienta escribe | las que escriben se marcan y pasan por aprobación (módulo 10.4) |

## Las tres herramientas del asistente

```
HERRAMIENTAS = [
 {"name":"buscar_documentos",
  "description":"Recupera fragmentos de documentos del cliente relevantes para una pregunta, con
     su cita. Respeta los permisos del usuario. No sirve para buscar un número exacto: use
     buscar_factura.",
  "parameters":{"type":"object","properties":{"pregunta":{"type":"string","maxLength":300},"
     coleccion":{"type":"string","enum":["facturas","contratos","actas","todas"]},"k":{"type":"
     integer","minimum":1,"maximum":8}},"required":["pregunta"]}},
 {"name":"buscar_factura",
  "description":"Devuelve UNA factura por su número exacto (FV-AAAA-NNNNNN). Si no existe,
     devuelve error no_encontrada.",
  "parameters":{"type":"object","properties":{"numero":{"type":"string","pattern":"^FV-\\d{4}-\\d
     {6}$"}},"required":["numero"]}},
 {"name":"crear_borrador_acta",
  "description":"ESCRIBE: crea un borrador de acta a partir de una transcripción. Requiere
     aprobación humana antes de guardarse. No la use para resumir: use responder.",
  "parameters":{"type":"object","properties":{"transcripcion_id":{"type":"string"},"plantilla":{"
     type":"string","enum":["comite","proveedor"]}},"required":["transcripcion_id","plantilla"]}}
]
```

## Poda de contexto en agentes largos

Un agente de diez pasos acumula diez resultados de herramientas. Si todos se quedan en la ventana, en el paso ocho ya no cabe la pregunta. La **poda** decide qué se queda:

| Estrategia | Qué hace | Cuándo |
|---|---|---|
| Resumen del resultado | cada resultado de herramienta se reemplaza por un resumen de 1–3 líneas en el paso siguiente | siempre |
| Estado fuera de la ventana | lo importante (campos ya extraídos, decisiones) va a un objeto de estado persistido; la ventana solo lleva un puntero | agentes de > 4 pasos |
| Ventana deslizante del historial | se conservan los últimos N turnos completos y un resumen de los anteriores | conversaciones largas |
| Recuperación en vez de memoria | el historial viejo se indexa y se recupera si hace falta | soporte de varios días |

La regla: **la ventana lleva lo que el modelo necesita para el siguiente paso**, no la historia completa. La historia completa vive en la traza (módulo 11) y en el estado (módulo 10).

> PROCEDIMIENTO · **Laboratorio 5.3**
>
> `labs/05-03/`: las tres herramientas con sus contratos y una función de poda que, dado un historial de pasos, devuelve la ventana presupuestada según 5.1. Pruebe con un historial sintético de 12 pasos y verifique que la ventana nunca pasa del tope.
> **Entregable:** `herramientas.json`, `podar.py` con prueba. Commit: «Capa 5.3: herramientas y poda».

### Qué se degrada en cada perfil

**A** — La poda es más agresiva: ventana de 4k–8k

**B** — Ventana de 8k–16k; resumen de resultados basta

**C** — Holgura; se sigue podando por costo de prefill

**Cuándo NO usar esto** **Diseño de herramientas y poda de contexto**

- No dé al modelo veinte herramientas: elige peor y gasta contexto en leerlas. Cinco a ocho por agente; si hacen falta más, es otro agente (módulo 10.2).
- No pode el historial de una conversación de **un** turno: no hay nada que podar.

PRÁCTICA · **Práctica**

1. Reescriba la descripción de `buscar_documentos` para que el modelo **no** la use cuando el usuario da un número de factura.

2. Un resultado de herramienta trae 4 000 tokens. Escriba la regla de resumen.

## Referencias

1. Anthropic (2024). *Building effective agents* — sección sobre diseño de herramientas.
2. OpenAI — guía de *function calling*: https://platform.openai.com/docs/guides/function-calling `[VOLÁTIL — verificado: 2026-09]`
3. Módulo 10 de este curso — estado persistido y presupuestos.