مختصر كتابي

من يختار الخطوة التالية؟



مَن ينبغي أن يختار الخطوة التالية: الكود أم الـModel؟

يراجع عمر، مدير العمليات في **شركة Seagull للتوزيع**، أسئلة مريم ويجد أن أكثرها تكرارًا هو سؤال المواعيد: «متى تصل طلباتي؟» (الطلبان `4800` و`4802` على حساب `C200`). لكل تشغيلة **الـTrace** الخاص بها الذي يسجّل كل خطوة ومَن اختارها.

على **الـAgent**، اختار **الـModel** الخطوات نفسها في 5 من 5 تشغيلات: قراءة الطلبين معًا ثم كتابة الرد. لكن لا ضمان لتكرار ذلك، لأن الـModel يقرّر كل مرة من البداية ومعه كل الـTools، بما فيها Tool ترسل رسالة مباشرة إلى الـCaller دون أن تراها مريم.

لذا بنينا **الـPipeline**: خطوات ثابتة ينفّذها الكود — يقرأ الطلبات بالـSession، ثم يقرأ ملف السياسة الموافق لحالة الطلب، ثم يستدعي الـModel مرة واحدة بلا Tools، ثم يمرّر الرد على الـGrounding Check. على سؤال المواعيد، خفّض الـPipeline طلبات الـModel من 2 إلى 1، وهبطت **الـTokens** الداخلة إلى نحو الربع (≈ 560 بدل ≈ 2,299).

لكن على سؤال مختلف في النسخة الأولى من الـPipeline — «أين الطلب `4802`، وإذا لم يصل بعد يريد الـCaller إلغاءه؟» — ظهر خللان من كودنا: أولًا لم تُقرأ سياسة الإلغاء (0 من 5)، فاقترح الرد في 3 من 5 تشغيلات تسجيل طلب إلغاء. ثانيًا قال الـModel في 5 من 5 تشغيلات إن النظام مضطرب؛ لأن الكود استخرج كل رقم فقرأ `200` من `C200` كطلب (`not_found`) بجانب `4802` (`dispatched`). أما الـAgent، ففي 5 من 5 تشغيلات قرأ الطلب وبحث عن سياسة الإلغاء معًا، وقال القاعدة الصحيحة (لا إلغاء بعد الشحن، بل مرتجع) كل مرة جلب البحث الملف (3 من 3).

الحل هو **الـPipeline** للأسئلة المعروفة مع **الـHandoff** إلى الـAgent لما يتجاوز حدودها. وبدل قائمة كلمات تفوّت الصياغات الجديدة، نعطي الـModel خيارًا واحدًا للخروج: كتابة `HANDOFF` وحدها، فيحوّل الكود السؤال إلى الـAgent دون أن تراها مريم. كما يلتقط الكود الأرقام بعد كلمة «طلب» فقط، ويحوّل فورًا إن غابت أو زادت على الحد، ويرفض عرض رد مقطوع أو فارغ. وتقرأ `who_decides` و`system_kind` الـTrace لتصنيف التشغيلة مما حدث فعليًا:

```
def who_decides(trace):
    out = {"code": 0, "model": 0}
    for step in trace if isinstance(trace, list) else []:
        if isinstance(step, dict) and step.get("kind") in ("tool", "handoff") and
step.get("decided_by") in out:
            out[step["decided_by"]] += 1
    return out

def system_kind(trace):
    steps = [s for s in trace if isinstance(s, dict)] if isinstance(trace, list)
else []
    actions = [s for s in steps if s.get("kind") == "tool"]
    models = [s for s in steps if s.get("kind") == "model"]
    if any(s.get("decided_by") != "code" for s in actions):
        return "agent"
    if actions:
        return "pipeline"
    if len(models) == 1 and not any(s.get("kind") == "handoff" for s in steps):
        return "call"
    return "unknown"
```

السؤال (5 تشغيلات، Claude Haiku 4.5)	الـAgent وحده	الـPipeline مع الـHandoff
سؤال المواعيد (4800 و4802)	2 طلب · ≈ 2,299 داخل · ≈ 0.00051$	1 طلب · ≈ 560 داخل · ≈ 0.00041$ · تحويل خطأ 0/5
سؤال الإلغاء (4802)	قرأ وبحث 5/5 · ≈ 0.00071$	تحوّل 5/5 · القاعدة صح 3/3 · ≈ 0.00089$

- في **الـAgent** يقرأ الـModel الرسالة ويختار الخطوة بعد رؤية النتيجة؛ وفي **الـPipeline** يقرأ الكود الرسالة وينفّذ خطوات ثابتة.
- استخراج الأرقام بعد كلمة «طلب» فقط يمنع الخلط مع `C200` وأرقام الهواتف والكميات.
- كلمة الخروج `HANDOFF` تتفوّق على قوائم الكلمات الثابتة مع الصياغات غير المتوقعة.
- أبسط الأنواع هو **الـCall** (استدعاء واحد للـModel بلا Tools ولا خطوات كود)، وتحدّد `system_kind` النوع مما حدث في الـTrace.
- ارتفاع تكلفة سؤال الإلغاء قليلًا ثمن مقبول مقابل ضمان خطوات السؤال المتكرّر وقابليته للـTests.

جرّب في ملفات المعمل (مجلد الدرس في Labs_00): أصلح استخراج أرقام الطلبات والـHandoff والتحقق من الرد في `Pipeline.run`، واكتب `who_decides` و`system_kind` لاجتياز الـTests الـ15 وتصنيف التشغيلات الـ35 المسجّلة.

الخلاصة: استخدم الـPipeline للأسئلة ذات الخطوات المعروفة، ودع الـHandoff يسلّم ما يتجاوز حدودها إلى الـAgent.