

Facsímil 10

Aprendizaje por refuerzo

Coleccionable completo · 8 capítulos

686f6c61 · 2026

Índice

01	MDP, políticas, retorno y Bellman
02	Datos de interacción: eventos, trayectorias y linaje
03	Exploración, bandits y validación de políticas
04	Offline RL y evaluación contrafactual de políticas
05	Preferencias y post-training: RLHF, DPO, RLAIIF y RLVR
06	Reward engineering: calidad de señal, verificadores y reward cards
07	Serving de políticas: monitorización, drift y cambios controlados
08	Recapitulación de aprendizaje por refuerzo

Capítulo 01: MDP, políticas, retorno y Bellman

La diferencia entre responder y decidir

Hasta ahora hemos evaluado modelos que predicen, clasifican, recuperan documentos, llaman herramientas o generan texto. El aprendizaje por refuerzo cambia la pregunta. Ya no preguntamos solo “¿la salida es correcta?”, sino “¿qué pasa después de actuar?”.

Un clasificador de tickets puede acertar una etiqueta. Un sistema de refuerzo decide qué hacer con el ticket, observa consecuencias y aprende qué política produce mejores resultados acumulados. Esa diferencia parece pequeña hasta que aparece el tiempo: cerrar rápido puede subir una métrica hoy y generar reaperturas mañana; pedir más información puede parecer lento y evitar errores posteriores; usar una herramienta cara puede resolver mejor, pero romper el presupuesto si se usa sin criterio.

En ingeniería, RL es útil aunque no entrenes un agente. Te da vocabulario para diseñar decisiones: estado, acción, recompensa, política, horizonte, valor futuro y efectos de optimizar una métrica incompleta. Ese vocabulario será clave para entender bandits, RLHF, RLAIIF, RLVR y agentes con evaluadores.

Sutton y Barto presentan el aprendizaje por refuerzo como el estudio de cómo un agente aprende a tomar acciones mediante interacción para maximizar una señal acumulada.¹ La idea es antigua, pero en IA moderna reaparece en post-training, routing, evaluación, agentes y sistemas que optimizan comportamiento.

El MDP como contrato de decisión

Un proceso de decisión de Markov (MDP, por *Markov decision process*) es el modelo estándar para describir decisiones que se toman una tras otra, cuando cada decisión cambia la situación y el resultado no es seguro. Es el marco formal con el que la teoría del refuerzo habla con precisión de estado, acción y consecuencia: en lugar de decir "el sistema decide y ya veremos qué pasa", un MDP obliga a declarar qué situaciones existen, qué puede hacer el agente en cada una, con qué probabilidad una acción lleva a la siguiente situación, qué recompensa deja y cuánto pesa el futuro frente al presente. Por eso lo tratamos como un contrato de decisión: no resuelve el problema, pero fija las piezas que cualquier método de refuerzo necesita para razonar.

Formalmente, un MDP se escribe:

$$\mathcal{M} = (S, A, P, R, \gamma)$$

SÍMBOL O	NOMBR E	QUÉ SIGNIFICA	EJEMPLO EN UN SISTEMA DE IA
S	Estados	Situaciones posibles.	Ticket nuevo, ticket con evidencia, ticket resuelto.
A	Acciones	Decisiones disponibles.	Responder, pedir datos, consultar RAG, escalar a persona.
$P(s' s, a)$	Transición	Probabilidad de pasar a s' al hacer a .	Si pido datos, quizá el usuario responde o abandona.
$R(s, a, s')$	Recompe nsa	Resultado numérico inmediato.	+2 si resuelve, -1 si usa tool cara, -5 si reabre.
γ	Descuent o	Peso del futuro.	$\gamma = 0.9$ valora consecuencias futuras; $\gamma = 0$ mira solo lo inmediato.

La palabra “Markov” significa que el estado actual contiene lo necesario para modelar el siguiente paso. No significa que el mundo sea simple. Significa que nosotros hemos decidido qué información metemos en el estado para que la predicción sea tratable.

En un producto real, esta es la parte más difícil. “Estado” no es una palabra elegante para “contexto”. Es una especificación. Si omites una variable importante, la política aprende con una fotografía incompleta.

ESTADO POBRE	QUÉ PIERDE	ESTADO MEJOR
ticket_abier to	No sabe prioridad, historial ni evidencia.	ticket_abierto + categoria + SLA + intentos + documentos_recuperados .
usuario	No distingue sesión, permiso ni consentimiento.	usuario + rol + permiso + politica_datos + canal .
respuesta_ge nerada	No sabe si hay cita ni si pasó eval.	respuesta_generada + cita + score_eval + coste + trazas .

Política y retorno

La política indica cómo actúa el sistema:

$$\pi(a | s) = P(A_t = a | S_t = s)$$

Puede ser determinista, si siempre elige la misma acción en un estado:

$$a = \pi(s)$$

o estocástica, si reparte probabilidad entre acciones:

$$\pi(\text{consultar_rag} \mid s) = 0.7, \quad \pi(\text{pedir_aclaracion} \mid s) = 0.3$$

El retorno desde el tiempo t es:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

PIEZA	LECTURA	EJEMPLO
R_{t+1}	Resultado inmediato.	Responder rápido.
γR_{t+2}	Consecuencia futura descontada.	El usuario confirma o reabre.
$\gamma^2 R_{t+3}$	Consecuencia aún más lejana.	El caso mejora la base de conocimiento.

Si $\gamma = 0$, el sistema solo valora la recompensa inmediata. Si γ se acerca a 1, el sistema valora más el futuro. En soporte, salud, educación o compliance, γ demasiado bajo produce políticas miopes. En sistemas de alto coste por acción, γ demasiado alto puede hacer que el sistema explore demasiado antes de dar valor.

Valor, Bellman y la idea de recursión

La función de valor de una política es:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

La función acción-valor es:

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

La diferencia entre V y Q es muy práctica. $V^\pi(s)$ te dice cuánto vale estar en un estado si sigues una política. $Q^\pi(s, a)$ te dice cuánto vale tomar una acción concreta desde ese estado y luego seguir la política. Si quieres elegir una acción, Q suele ser más directo.

Bellman convierte esa intuición en una ecuación recursiva. Richard Bellman formuló la programación dinámica como una forma de resolver problemas secuenciales dividiéndolos en subproblemas conectados por valor futuro.² En un MDP, esa idea se lee así:

$$V^\pi(s) = \sum_a \pi(a | s) \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V^\pi(s'))$$

No te quedes en la fórmula. Léela como una frase:

El valor de estar en s es el promedio, bajo la política, de recompensa inmediata más valor futuro esperado.

TÉRMINO	QUÉ HACE	SI LO OLVIDAS
$\pi(a s)$	Pondera qué acciones toma la política.	Evalúas una política que no se parece a la que ejecutas.
$P(s' s, a)$	Modela consecuencias.	Tratas el mundo como si no respondiera.
$R(s, a, s')$	Define qué cuenta como buen resultado.	Optimizas una señal cómoda, no el objetivo.
$\gamma V^\pi(s')$	Mete futuro.	El sistema se queda corto de horizonte.

El matiz importante: Bellman no es una sola fórmula. Es una familia de ecuaciones. Puedes usarla para **evaluar una política** o para **buscar una política mejor**.

Bellman óptimo: cuando queremos decidir mejor

Si queremos la mejor política posible dentro del MDP definido, escribimos valor óptimo:

$$V^*(s) = \max_a \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V^*(s'))$$

Y acción-valor óptimo:

$$Q^*(s, a) = \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma \max_{a'} Q^*(s', a'))$$

SÍMBOLO	SIGNIFICADO	LECTURA CERCANA
$V^*(s)$	Mejor valor posible desde s .	Si estoy aquí, cuánto puedo esperar como máximo.
$Q^*(s, a)$	Mejor valor posible tras tomar a en s .	Si hago esto ahora, cuánto puedo esperar después.
$\max_a$	Elige la acción con más retorno esperado.	No elige la más bonita, elige la que mejor futuro modelado produce.
$\max_{a'} Q^*(s', a')$	Mejor acción futura desde el nuevo estado.	Lo de ahora se evalúa mirando qué abre después.

Puterman trata los MDP como el marco matemático estándar para decisiones secuenciales bajo incertidumbre, con políticas, transiciones, recompensas y métodos de programación dinámica.³ Esa palabra, dinámica, importa: el valor de una decisión no está solo en la acción actual, sino en el estado que deja preparado.

Algoritmos fundamentales

Una vez tienes S, A, P, R, γ , aparecen varios caminos. No todos significan lo mismo.

Howard formuló la iteración de políticas como una forma práctica de alternar dos preguntas: cuánto vale la política actual y qué política resulta si actuamos de forma codiciosa respecto a ese valor.⁴ Bertsekas desarrolla esta familia de métodos como el núcleo de la programación dinámica moderna: resolver una decisión grande mediante ecuaciones locales de valor, convergencia y mejora iterativa.⁵

ALGORITMO	QUÉ NECESITA	QUÉ PRODUCE	CUÁNDO ENCAJA
Evaluación de política	Política fija π , transición y recompensa.	V^π .	Quieres saber cuánto vale una política ya definida.
Iteración de valor	Modelo P, R .	Aproximación de V^* .	Tienes modelo del entorno y quieres política óptima.
Iteración de políticas	Modelo P, R y una política inicial.	Políticas cada vez mejores.	Puedes alternar evaluar y mejorar.
Q-learning	Experiencias (s, a, r, s') .	Tabla o función Q .	No conoces P , pero observas interacción.
Deep Q-Network	Experiencias y red neuronal.	Aproximación de Q en estados grandes.	El espacio de estados no cabe en una tabla.

La evaluación iterativa de una política aplica:

$$V_{k+1}^\pi(s) = \sum_a \pi(a | s) \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V_k^\pi(s'))$$

La iteración de valor aplica:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V_k(s'))$$

La mejora de política elige:

$$\pi_{\text{nueva}}(s) = \arg \max_a \sum_{s'} P(s' | s, a) (R(s, a, s') + \gamma V^\pi(s'))$$

Y Q-learning actualiza con experiencia observada:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t) \right)$$

SÍMBOLO	QUÉ SIGNIFICA	EJEMPLO
α	Tasa de aprendizaje.	0,1: actualiza poco; 0,8: actualiza mucho.
r_{t+1}	Recompensa observada.	+2 si la respuesta se acepta.
$\gamma \max_{a'} Q_t(s_{t+1}, a')$	Mejor futuro estimado desde el nuevo estado.	Qué podríamos conseguir después.
$r + \gamma \max Q - Q$	Error temporal.	Diferencia entre lo esperado y lo observado.

Watkins y Dayan formalizaron Q-learning como un método que aprende valores acción-estado a partir de experiencia, sin requerir conocer de antemano las probabilidades de transición.⁶ Mucho después, Deep Q-Networks conectaron Q-learning con redes neuronales para aproximar Q en espacios grandes, como juegos de Atari con píxeles como entrada.⁷

Para el facsímil, la idea no es memorizar algoritmos. La idea es distinguir preguntas:

SI PREGUNTAS...	ESTÁS HACIENDO...
“¿Cuánto vale esta política?”	Evaluación de política.
“¿Qué acción maximiza futuro si conozco el entorno?”	Iteración de valor o políticas.
“¿Qué aprendo de cada experiencia observada?”	Q-learning o variantes.
“¿Puedo evaluar otra política con datos históricos?”	Offline RL y evaluación contrafactual, capítulo 04.

Ejemplo numérico con transición

Volvamos al asistente de soporte. Queremos decidir qué hacer en s_0 : ticket nuevo con información incompleta.

ESTADO	SIGNIFICADO	VALOR ESTIMADO
s_0	Ticket nuevo incompleto.	Lo calcularemos.
s_1	Ticket con evidencia suficiente.	$V(s_1) = 4$.
s_2	Ticket resuelto.	$V(s_2) = 3$.
s_3	Ticket reabierto.	$V(s_3) = -1$.

Acciones disponibles:

ACCIÓN	RECOMPENSA INMEDIATA	TRANSICIÓN
a_p : pedir aclaración	-0.2	80 % a s_1 , 20 % sigue en s_0 con valor aproximado 1.
a_r : responder ya	+1.0	55 % a s_2 , 45 % a s_3 .

Con $\gamma = 0.9$:

$$Q(s_0, a_p) = -0.2 + 0.9(0.8 \cdot 4 + 0.2 \cdot 1) = 2.86$$

$$Q(s_0, a_r) = 1 + 0.9(0.55 \cdot 3 + 0.45 \cdot (-1)) = 2.08$$

La acción a_r parece más atractiva si miras solo la recompensa inmediata: +1.0 frente a -0.2. Pero el retorno esperado dice otra cosa. Pedir aclaración abre un estado con más evidencia, y ese futuro compensa el coste inicial.

Esta es la intuición que quiero que te lleves. Bellman obliga a dejar de discutir “qué acción suena mejor” y empezar a discutir “qué futuro medible abre esta acción”.

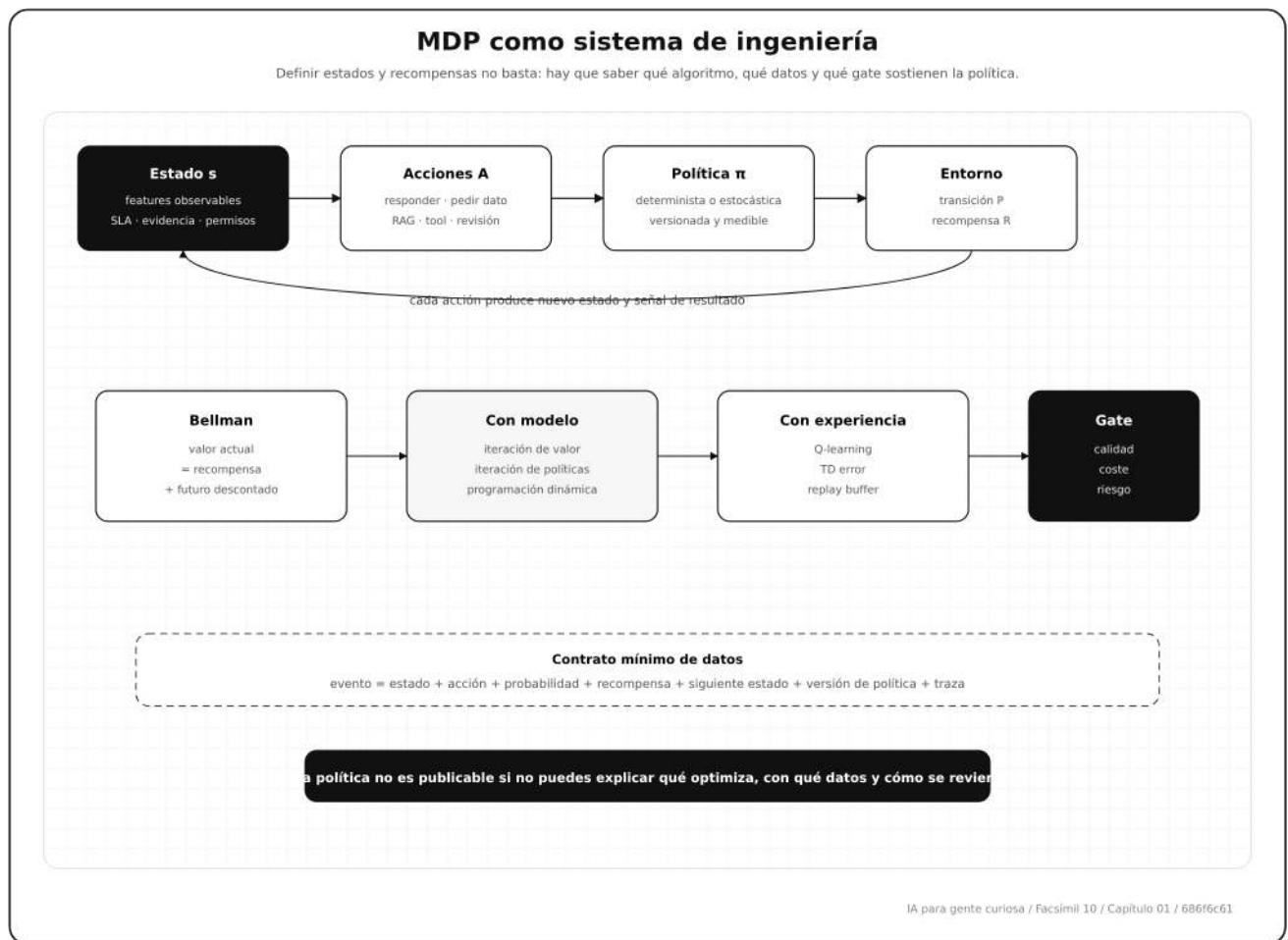
Modelo conocido, modelo aprendido y datos

No todos los problemas RL se resuelven igual. La división más importante para ingeniería es si conoces el modelo del entorno.

CASO	QUÉ TIENES	QUÉ HACES
Modelo conocido	$P(s' s, a)$ y $R(s, a, s')$.	Programación dinámica, iteración de valor, iteración de políticas.
Modelo simulado	Un simulador que genera transiciones.	Entrenas y evalúas en entorno controlado.
Modelo desconocido con interacción	Eventos reales (s, a, r, s') .	Q-learning, bandits, métodos basados en experiencia.
Solo datos históricos	Logs de políticas anteriores.	Offline RL y evaluación contrafactual.
Preferencias humanas o verificables	Pares elegido/rechazado, checkers o rúbricas.	Post-training y diseño de recompensas.

Este cuadro conecta el capítulo 01 con el capítulo 02. Antes de hablar de datos de interacción, necesitamos saber qué debe contener cada dato: estado, acción, recompensa, siguiente estado y versión de política. Sin esa estructura, los algoritmos del cuadro se quedan en teoría.

Anatomía ampliada de un MDP de ingeniería

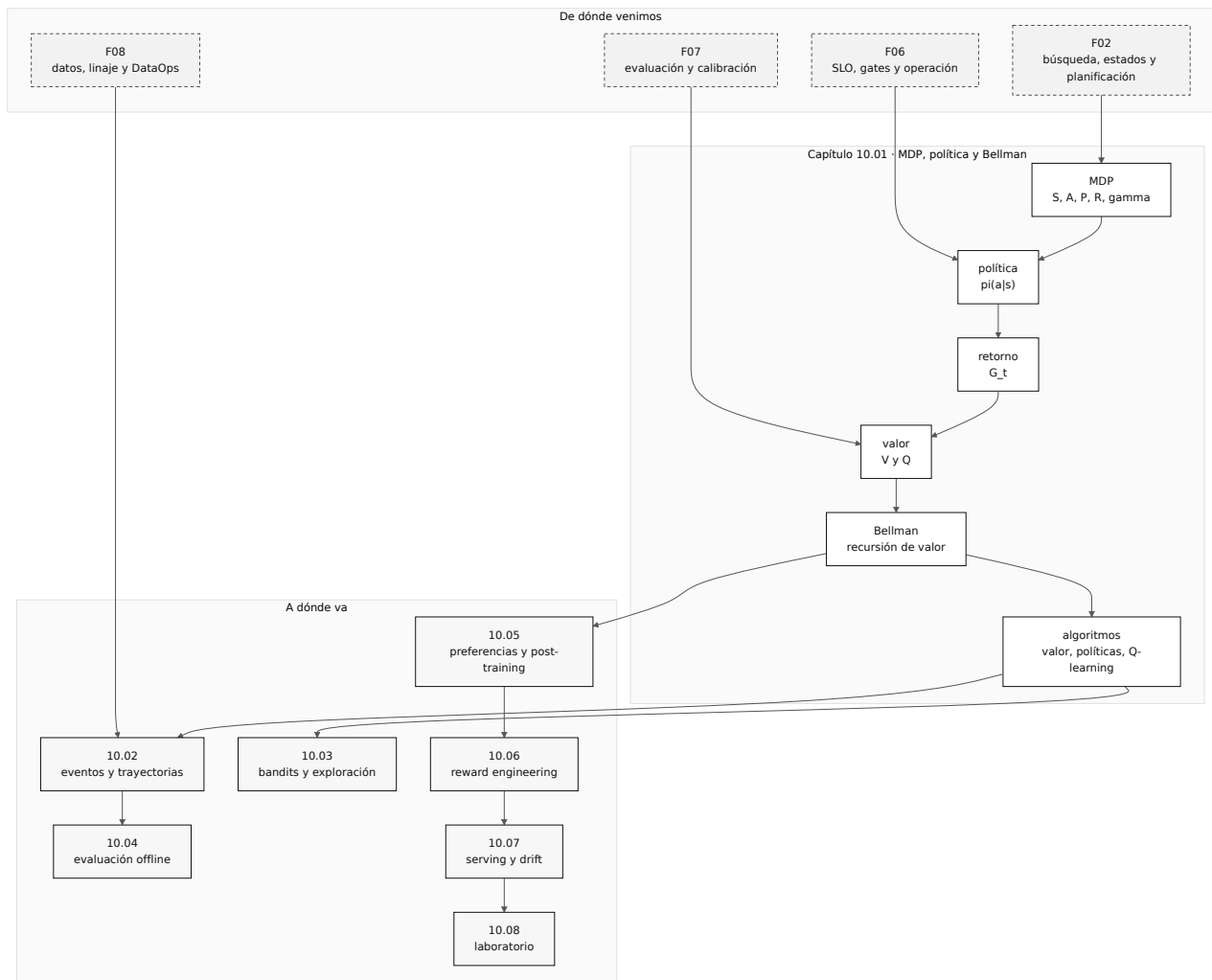


Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Llamar RL a cualquier sistema que decide	La palabra “agente” seduce.	Preguntar si hay estado, acción, transición, recompensa y aprendizaje.
Confundir contexto con estado	En LLMs todo parece contexto.	Definir qué variables hacen falta para predecir consecuencias.
Optimizar recompensa inmediata	Es fácil de medir.	Escribir retorno y decidir γ .
Tratar Bellman como fórmula decorativa	Parece abstracta.	Leerla como “ahora más futuro” y aplicarla a un ejemplo.
Saltar a Q-learning sin contrato de datos	El algoritmo parece el centro.	Registrar estado, acción, recompensa, siguiente estado y versión.
No separar política de evaluación	Se evalúa una cosa y se ejecuta otra.	Versionar política, reglas, prompts, rewards y evaluadores.

Cómo encaja todo

Este mapa rehace el capítulo como parte de todo el facsímil. Heredamos estados y planificación de IA clásica, añadimos Bellman y algoritmos, y preparamos los datos, la exploración, la evaluación offline, el post-training y la operación de políticas.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
MDP	Modelo formal de decisión con estados, acciones, transiciones, recompensas y descuento.
Política	Regla o distribución que decide acciones desde estados.
Retorno	Suma de recompensas futuras descontadas.
Valor V	Retorno esperado desde un estado.
Valor Q	Retorno esperado desde un estado y una acción concreta.
Bellman	Recursión que conecta valor actual con recompensa inmediata y valor futuro.
Iteración de valor	Algoritmo que actualiza valores usando el máximo sobre acciones.
Iteración de políticas	Alternancia entre evaluar una política y mejorarla.
Q-learning	Actualización de valores acción-estado a partir de experiencia observada.

Antes de pasar página

- ¿Puedes escribir un MDP como (S, A, P, R, γ) ?
- ¿Qué información debe estar en el estado para que una política de soporte no sea miope?
- ¿Qué significa $\pi(a | s)$?
- ¿Qué cambia si $\gamma = 0$ frente a $\gamma = 0.95$?
- ¿Por qué $Q(s, a)$ puede ser más útil que $V(s)$ para elegir una acción?
- ¿Qué diferencia hay entre evaluar una política y buscar una política óptima?
- ¿Cuándo usarías iteración de valor y cuándo Q-learning?
- ¿Qué artefacto dejarías para demostrar qué política se ejecutó?

Para saber más

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control* (Vol. 2, 4.^a ed.). Athena Scientific.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. y Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529-533.
<https://doi.org/10.1038/nature14236>

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons. <https://doi.org/10.1002/9780470316887>

Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.^a ed.). MIT Press. <https://incompleteideas.net/book/the-book-2nd.html>

Watkins, C. J. C. H. y Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279-292.
<https://doi.org/10.1007/BF00992698>

En resumen

IDEA	QUÉ DEBES RECORDAR
RL empieza con una decisión formal.	Sin S, A, P, R, γ , puede haber automatización, pero no un MDP claro.
Bellman mete futuro en la decisión.	No evalúa solo la recompensa inmediata.
V y Q responden preguntas distintas.	V valora estados; Q ayuda a elegir acciones.
Los algoritmos dependen de lo que sabes.	Si conoces P, R , usa programación dinámica; si observas experiencia, necesitas datos.
El capítulo 02 se vuelve inevitable.	Todo algoritmo serio acaba pidiendo eventos, trayectorias y linaje.

Notas

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.
<https://mitpress.mit.edu/9780262039246/reinforcement-learning/>. Consultado el 7 de junio de 2026. ↵
2. Bellman, R. (1957). *Dynamic Programming*. Princeton University Press. ↵
3. Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons. <https://doi.org/10.1002/9780470316887>. ↵
4. Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press. ↵
5. Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control* (Vol. 2, 4.^a ed.). Athena Scientific. ↵
6. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279-292.
<https://doi.org/10.1007/BF00992698>. ↵
7. Mnih, V. y otros (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529-533. <https://doi.org/10.1038/nature14236>. ↵

Cuadernos para practicar

Escanea el código con el móvil para abrir el cuaderno en Google Colab y ejecutarlo sin instalar nada.



Aprender a decidir: MDP, Bellman y un mundo de casillas

<https://colab.research.google.com/github/686f6c61/iaparagentecuriosa-notebooks/blob/main/Facs%C3%ADmil%2010/01-mdp-gridworld-bellman.ipynb>

Capítulo 02: Datos de interacción: eventos, trayectorias y linaje

Antes de aprender, hay que registrar bien

El capítulo anterior nos dio el lenguaje matemático: estados, acciones, transiciones, recompensas, políticas y retorno. Eso sirve para pensar. Pero un sistema real no aprende de una definición bonita. Aprende de datos que alguien ha capturado, validado, versionado y defendido.

Aquí cambia la mentalidad. En un problema supervisado puedes empezar con una tabla: entrada, etiqueta y metadatos. En RAG puedes empezar con documentos, chunks, fuentes y versión. En aprendizaje por refuerzo, una fila aislada casi nunca basta, porque la acción de hoy modifica la observación de mañana. El dato tiene tiempo, política, decisión, consecuencia y linaje.

Si lo miramos como ingenieros de datos, la pregunta importante no es todavía “qué algoritmo entrenamos”. La pregunta es:

¿podemos reconstruir qué vio la política, qué opciones tenía, qué eligió, con qué probabilidad, qué ocurrió después y con qué versión de sistema se produjo todo?

Sutton y Barto describen el aprendizaje por refuerzo como aprendizaje mediante interacción entre agente y entorno.¹ En ingeniería, esa palabra, interacción, se traduce en eventos, trayectorias, contratos de datos, metadatos y snapshots reproducibles.

Qué no es un dataset de refuerzo

Un dataset de refuerzo no es una colección de prompts con respuestas. Tampoco es un log de aplicación con mensajes de depuración. Y no es una lista de recompensas sueltas. Esas piezas pueden ayudar, pero no bastan para saber si una política produjo una buena decisión.

El fallo típico consiste en guardar lo que era cómodo para depurar, no lo que hará falta para evaluar. Guardamos la respuesta final, pero no las acciones disponibles. Guardamos una recompensa, pero no cuándo llegó. Guardamos el modelo usado, pero no la versión de política que eligió la acción. Guardamos trazas técnicas, pero no la probabilidad con la que se tomó la decisión.

ERROR DE DATOS	QUÉ PARECE QUE TIENES	QUÉ FALTA PARA RL
Solo guardar la respuesta final.	Un historial legible de interacciones.	Estado, acción, alternativas, política, propensión y consecuencia.
Guardar recompensa sin <code>reward_time</code> .	Una métrica de resultado.	Atribución temporal: a qué acción se asocia esa señal.
Mezclar versiones de política.	Más volumen de datos.	Comparabilidad y reproducibilidad.
Omitir <code>available_actions</code> .	La acción tomada.	El conjunto de decisiones posibles en ese estado.
Omitir <code>action_probability</code> .	Qué eligió la política.	Con qué probabilidad lo eligió, clave para evaluación offline.
No separar <code>event_time</code> e <code>ingestion_time</code> .	Un timestamp.	Orden real del mundo frente a orden de llegada al pipeline.
No registrar contrato ni hash del snapshot.	Una carpeta de datos.	Evidencia de qué se entrenó o evaluó exactamente.

Sculley y coautores explicaron que en sistemas ML la deuda técnica aparece en datos, dependencias, configuración, cambios silenciosos y bucles de retroalimentación.² En RL esa deuda se multiplica: la política que toma decisiones también decide qué datos veremos después.

Del log malo al evento defendible

Una forma rápida de entender el salto es comparar un log de aplicación con un evento de decisión. El log de aplicación sirve para depurar. El evento RL sirve para reconstruir, evaluar y aprender.

REGISTRO POBRE	POR QUÉ NO BASTA	EVENTO DEFENDIBLE
2026-06-07 09:00 usuario pide ayuda	No dice qué vio la política ni qué podía hacer.	state_id , state_features , available_actions .
accion=pedir_dato	No dice si esa acción era una opción razonable o la única disponible.	action , available_actions , policy_version .
modelo=gpt-x	El modelo no es la política completa. Falta routing, prompt, reglas y umbrales.	policy_id , policy_version , environment_version .
score=0.82	Score de qué, calculado cuándo y con qué versión.	reward , reward_terms , reward_version , reward_time .
ok=true	No separa éxito inmediato, reapertura, coste ni groundedness.	Componentes de recompensa separados.
trace=abc	La traza ayuda, pero no sustituye el contrato de decisión.	trace_id más evento RL completo.

Ejemplo de log pobre:

```
2026-06-07T09:00:00Z case_001 accion=pedir_dato ok=false modelo=asistente-v4
```

Ejemplo de evento defendible, abreviado:

```
{
  "episode_id": "case_001",
  "t": 0,
  "state_id": "ticket_nuevo",
  "available_actions": ["consultar_rag", "pedir_dato", "escalar_revision"],
  "action": "pedir_dato",
  "action_probability": 0.34,
  "policy_version": "2026-06-07.3",
  "reward": -0.2,
  "reward_version": "support_reward.v2",
  "reward_time": "2026-06-07T09:04:00Z",
  "next_state_id": "esperando_usuario"
}
```

La diferencia no es estética. El primer registro permite contar una historia aproximada. El segundo permite escribir tests, calcular retorno, medir cobertura, comparar políticas y decidir si un snapshot puede usarse.

El evento mínimo que sí permite reconstruir una decisión

Un evento de interacción es el registro atómico de una decisión. No debe intentar contar toda la historia del usuario ni guardar datos innecesarios. Debe guardar lo suficiente para reconstruir el paso t de una trayectoria.

Este sería un evento razonable para un asistente que decide si responder, pedir más datos o escalar a revisión:

```
{
  "schema_version": "rl_event.v1",
  "event_id": "evt_2026_06_07_00042",
  "episode_id": "case_matricula_731",
  "t": 3,
  "event_time": "2026-06-07T12:05:31Z",
  "ingestion_time": "2026-06-07T12:05:34Z",
  "reward_time": "2026-06-07T12:22:10Z",
  "state_id": "ticket_con_evidencia",
  "state_features": {
    "categoria": "matricula",
    "sla_horas_restantes": 18,
    "evidencias_recuperadas": 2,
    "confianza_eval": 0.82
  },
  "available_actions": ["responder", "pedir_dato", "escalar_revision"],
  "action": "pedir_dato",
  "action_probability": 0.31,
  "policy_id": "routing_policy_uch",
  "policy_version": "2026-06-07.3",
  "reward": -0.2,
  "reward_version": "support_reward.v2",
  "reward_terms": {
    "resolved": 0.0,
    "groundedness": 0.9,
    "cost": -0.2,
    "reopened": 0.0
  },
  "next_state_id": "esperando_usuario",
  "terminal": false,
  "trace_id": "trace_9be2",
  "environment_version": "support_runtime.2026-06-07",
  "data_minimization": {
    "contains_personal_data": false,
    "redaction_policy": "support_redaction.v1"
  }
}
```

Cada campo existe por una razón técnica:

CAMPO	QUÉ PERMITE DEFENDER
<code>schema_version</code>	Qué contrato de datos aplica y cómo migrar versiones.
<code>event_id</code>	Idempotencia, deduplicación y trazabilidad.
<code>episode_id</code>	Agrupar eventos en una trayectoria.
<code>t</code>	Orden lógico dentro del episodio.
<code>event_time</code>	Cuándo ocurrió la decisión en el mundo.
<code>ingestion_time</code>	Cuándo llegó al pipeline.
<code>reward_time</code>	Cuándo se observó la consecuencia.
<code>state_id</code> y <code>state_features</code>	Qué información vio la política.
<code>available_actions</code>	Qué podía elegir legalmente en ese estado.
<code>action</code>	Qué eligió.
<code>action_probability</code>	Propensión de la política histórica.
<code>policy_id</code> y <code>policy_version</code>	Qué lógica produjo la acción.
<code>reward</code> y <code>reward_terms</code>	Resultado total y componentes auditables.
<code>reward_version</code>	Qué definición de recompensa se usó.
<code>next_state_id</code>	Transición observada.
<code>terminal</code>	Si el episodio termina ahí.
<code>trace_id</code>	Unión con logs, spans, latencia, coste y errores.
<code>environment_version</code>	UI, permisos, catálogo, datos externos y runtime que condicionaron la decisión.
<code>data_minimization</code>	Evidencia de privacidad y minimización.

Baylor y coautores describieron TFX como una plataforma donde datos, transformaciones, validación, modelos y metadatos forman un sistema de producción, no una cadena informal de scripts.³ Un evento RL debe vivir en esa misma filosofía: datos, contrato, validación, metadatos y linaje.

Contrato de datos: no basta con un JSON bonito

Un evento de ejemplo enseña la forma. Un contrato obliga al sistema a cumplirla. En una arquitectura seria, el contrato define campos obligatorios, tipos, catálogos, rangos, claves, reglas temporales, evolución de versiones y criterios de rechazo.

Fragmento simplificado de contrato:

```
{
  "schema_version": "rl_event.v1",
  "primary_key": ["event_id"],
  "episode_key": ["episode_id"],
  "required": [
    "event_id",
    "episode_id",
    "t",
    "event_time",
    "ingestion_time",
    "state_id",
    "available_actions",
    "action",
    "action_probability",
    "policy_version",
    "reward",
    "reward_version",
    "next_state_id",
    "terminal",
    "trace_id"
  ],
  "constraints": {
    "action_in_available_actions": true,
    "probability_range": [0.0, 1.0],
    "event_time_before_ingestion_time": true,
    "reward_time_after_event_time": true,
    "terminal_event_closes_episode": true
  }
}
```

Ese contrato no es burocracia. Es el equivalente, en datos, a una interfaz de API. Si cambia sin control, rompe consumidores. Si permite campos ambiguos, genera datasets que no se pueden comparar. Si no define reglas de evolución, una política entrenada en junio quizá no sea reproducible en julio.

Breck y coautores propusieron el ML Test Score como una rúbrica para medir preparación de sistemas ML en producción, incluyendo pruebas de datos, infraestructura, monitorización y reproducibilidad.⁴ El contrato de evento RL es una pieza de esa preparación: antes de entrenar o evaluar, comprobamos si los datos merecen ser usados.

Trayectorias: una fila no cuenta la historia

Una trayectoria o episodio es una secuencia ordenada de decisiones:

$$\tau = (s_0, a_0, p_0, r_1, s_1, a_1, p_1, r_2, \dots, s_T)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
τ	Trayectoria completa.	Caso de soporte desde apertura hasta cierre.
s_t	Estado en el paso t .	Ticket con evidencia recuperada.
a_t	Acción tomada.	Pedir dato, responder, escalar.
p_t	Propensión de la acción observada.	La política eligió <code>pedir_dato</code> con 0,31.
r_{t+1}	Recompensa observada después de actuar.	+2 si se resuelve, -1 si se reabre.
T	Longitud del episodio.	4 pasos antes de terminar.

El retorno de una trayectoria es:

$$G(\tau) = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$$

SÍMBOLO	SIGNIFICADO	VALOR DE EJEMPLO
$G(\tau)$	Retorno acumulado descontado.	2,349
γ	Descuento del futuro.	0,9
r_{t+1}	Recompensa tras la acción del paso t .	-0,1; -0,2; +2,0; +1,0
T	Número de transiciones con recompensa.	4

Ejemplo:

PASO	ACCIÓN	RECOMPENSA
0	consultar RAG	-0,1
1	pedir dato	-0,2
2	responder con cita	+2,0
3	cierre sin reapertura	+1,0

Con $\gamma = 0.9$:

$$G(\tau) = -0.1 + 0.9(-0.2) + 0.9^2(2.0) + 0.9^3(1.0) = 2.349$$

Si guardas solo la última respuesta, no puedes calcular esto. Si guardas solo el coste, tampoco. Si guardas solo la recompensa final, pierdes qué acciones abrieron o cerraron el camino. La trayectoria es la unidad natural cuando el sistema decide varias veces antes de observar el resultado.

Recompensas tardías y ventanas de atribución

En muchas aplicaciones, la recompensa no aparece en el mismo instante que la acción. Un usuario puede reabrir un ticket horas después. Un alumno puede valorar una explicación al final de la práctica. Una recomendación puede producir lectura hoy y abandono mañana. Un agente puede resolver una tarea y descubrir más tarde que la tool usada era innecesaria.

Por eso separamos tres tiempos:

TIEMPO	QUÉ REPRESENTA	ERROR TÍPICO SI LO MEZCLAS
event_time	Momento real de la decisión.	Ordenas por llegada al pipeline y cambias la historia.
ingestion_time	Momento en que el evento entra al sistema de datos.	Confundes retraso de infraestructura con comportamiento del usuario.
reward_time	Momento en que la consecuencia se observa o se calcula.	Atribuyes una señal tardía a una acción equivocada.

La atribución de recompensa se puede escribir como una función explícita de sus ingredientes. No es un estándar universal, sino una forma de recordar que la recompensa depende de la observación posterior, de la acción evaluada, del estado previo y de una ventana temporal explícita:

$$r_{t+1} = f(o_{t+1}, a_t, s_t, \Delta t \leq W)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
r_{t+1}	Recompensa atribuida al paso t .	+2 por resolución confirmada.
o_{t+1}	Observación posterior.	Confirmación, reapertura, coste, revisión.
a_t	Acción que queremos evaluar.	Responder con cita.
s_t	Estado previo.	Ticket con evidencia.
Δt	Tiempo entre acción y observación.	17 minutos.
W	Ventana máxima de atribución.	48 horas.

La ventana W no es un detalle menor. Si es demasiado corta, no capturas consecuencias reales. Si es demasiado larga, mezclas efectos de muchas decisiones. En el contrato de datos debe aparecer cómo se atribuye cada recompensa, qué ocurre con episodios abiertos y cuándo un snapshot se considera maduro.

Arquitectura de datos: bronze, silver y gold para RL

Un pipeline RL defendible se parece más a una plataforma de datos que a una carpeta de experimentos. La estructura bronze, silver y gold ayuda a separar responsabilidades:

CA PA	QUÉ CONTIENE	QUÉ NO DEBERÍA CONTENER	EJEMPLO DE ARTEFACTO
Bronze	Eventos crudos recibidos del producto.	Decisiones de entrenamiento.	<code>raw_rl_events/2026-06-07/*.jsonl</code>
Silver	Eventos validados, deduplicados y enriquecidos.	Eventos que violan contrato.	<code>validated_events_v1</code>
Gold	Trayectorias, snapshots y datasets listos para evaluación.	Filas sin linaje o sin criterios de inclusión.	<code>replay_snapshot_2026_06_07_sha256...</code>

El replay buffer no es solo una tabla grande. Es un producto de datos. Debe responder:

PREGUNTA	POR QUÉ IMPORTA
¿Qué eventos entran y cuáles se rechazan?	Evita entrenar con datos rotos.
¿Qué política produjo cada evento?	Permite comparar y reproducir.
¿Qué reward se usó?	Evita mezclar objetivos distintos.
¿Qué distribución de estados y acciones cubre?	Detecta zonas sin evidencia.
¿Qué datos se minimizan o se eliminan?	Reduce exposición innecesaria.
¿Qué hash identifica el snapshot?	Permite repetir entrenamiento o evaluación.
¿Qué contrato lo validó?	Permite auditar cambios de schema.

TensorFlow ML Metadata modela artefactos, ejecuciones, contextos y linaje dentro de pipelines ML.⁵ OpenLineage propone un estándar abierto para representar jobs, datasets, runs y relaciones de linaje.⁶ El principio que nos interesa es estable: un dataset de interacción no debe ser anónimo; debe decir de dónde viene, cómo se procesó y qué versión exacta se usó.

Propensión: el campo que se echa de menos cuando ya es tarde

La propensión es la probabilidad con la que la política que generó el dato eligió la acción registrada:

$$p_t = \pi_b(a_t \mid s_t)$$

SÍMBOLO	SIGNIFICADO
p_t	Probabilidad registrada de la acción tomada.
π_b	Política de comportamiento, la que produjo el dato histórico.
a_t	Acción observada.
s_t	Estado observado por la política.

¿Por qué nos importa tanto? Porque después quizá queramos estimar qué habría pasado con otra política π_e sin desplegarla todavía. Una forma básica de ponderación por importancia usa:

$$w_t = \frac{\pi_e(a_t | s_t)}{\pi_b(a_t | s_t)}$$

SÍMBOLO LECTURA

w_t	Peso que corrige cuánto se parece la política nueva a la histórica.
π_e	Política que queremos evaluar.
π_b	Política que produjo el dato.

Para una trayectoria completa, una versión simplificada de importancia acumulada sería:

$$\widehat{V_{IS}(\pi_e)} = \frac{1}{n} \sum_{i=1}^n \left(\prod_{t=0}^{T_i-1} \frac{\pi_e(a_{i,t} | s_{i,t})}{\pi_b(a_{i,t} | s_{i,t})} \right) G(\tau_i)$$

SÍMBOLO SIGNIFICADO

$V_{IS}(\pi_e)$	Estimación offline del valor de la política nueva.
$\widehat{n}$	Número de trayectorias históricas.
T_i	Longitud de la trayectoria i .
$G(\tau_i)$	Retorno observado de la trayectoria.
π_e/π_b	Corrección entre política evaluada y política histórica.

La fórmula no es una invitación a usarla a ciegas. Puede tener varianza alta si las políticas se parecen poco o si algunas propensiones son muy pequeñas. La idea importante para este capítulo es más básica: si no registraste $\pi_b(a_t | s_t)$, no puedes aplicar muchas técnicas de evaluación offline con rigor. Dudík, Langford y Li desarrollaron estimadores doblemente robustos para evaluación y aprendizaje de políticas combinando información de propensión y modelos de recompensa.⁷ Li y coautores muestran esta lógica en recomendación de noticias con contextual bandits y datos registrados de políticas anteriores.⁸

Cobertura: qué estados y acciones sí has observado

Un replay buffer puede tener millones de eventos y seguir siendo pobre. Si casi todos los eventos vienen de estados fáciles, o si una acción apenas se ha probado, el volumen global engaña.

Una forma sencilla de medir cobertura estado-acción es:

$$c(s, a) = \frac{1}{N} \sum_{i=1}^N \mathbf{1}\{s_i = s, a_i = a\}$$

SÍMBOL O	SIGNIFICADO	EJEMPLO
$c(s, a)$	Cobertura empírica de una pareja estado-acción.	0,08
N	Número total de eventos.	10.000
$\mathbf{1}\{.\}$	Indicador: vale 1 si se cumple la condición.	1 si <code>ticket_con_evidencia</code> y <code>responder</code> .
s_i, a_i	Estado y acción del evento i .	<code>ticket_nuevo</code> , <code>pedir_dato</code> .

La cobertura no decide sola, pero guía la confianza:

SEÑAL	QUÉ SIGNIFICA	DECISIÓN PRUDENTE
Mucha cobertura en estados frecuentes.	Hay evidencia para decisiones comunes.	Evaluar offline y preparar piloto controlado.
Cero cobertura en una acción permitida.	No sabemos cómo se comporta.	Explorar primero o mantener revisión.
Cobertura desbalanceada por canal.	La política histórica casi no vio algunos casos.	Medir slices antes de automatizar.
Propensiones muy pequeñas.	Ponderaciones inestables.	Recortar pesos, usar estimadores robustos o no concluir.
Cambios de entorno sin versión.	Los datos mezclan mundos distintos.	Separar snapshots por <code>environment_version</code> .

Great Expectations y Pandera representan dos formas habituales de validar expectativas y schemas de datos en proyectos de datos y ML.⁹¹⁰ En RL, además de tipos y rangos, hay que validar cobertura temporal, cobertura de acciones y coherencia de trayectorias.

Tres ejemplos que sí se parecen a trabajo real

Veamos cómo cambia el contrato según el sistema.

Soporte con IA

El sistema decide si responder, pedir más datos, consultar RAG o escalar a revisión. El estado incluye categoría, SLA, evidencias recuperadas, confianza de evaluación y permisos. La recompensa puede mezclar resolución, groundedness, coste y reapertura.

CAMPO CRÍTICO	POR QUÉ IMPORTA
<code>available_actions</code>	Puede que <code>responder</code> no esté permitido sin evidencia suficiente.
<code>reward_time</code>	La reapertura puede llegar horas después.
<code>trace_id</code>	Permite saber si RAG falló, si una tool fue lenta o si faltó cita.
<code>reward_terms.reopened</code>	Separa resolver ahora de generar trabajo después.

Recomendador de contenidos

El sistema decide qué documento, noticia o recurso mostrar. El estado incluye usuario anonimizado, contexto, disponibilidad de contenido y restricciones. La recompensa puede ser lectura útil, permanencia, feedback explícito o continuación de tarea.

CAMPO CRÍTICO	POR QUÉ IMPORTA
<code>action_probability</code>	Sin propensión, no puedes comparar con rigor otra política de ranking.
<code>available_actions</code>	El catálogo visible cambia con permisos, idioma y disponibilidad.
<code>environment_version</code>	Un cambio de UI altera el comportamiento observado.
<code>data_minimization</code>	No todo atributo de usuario debe entrar en estado.

Mini evento abreviado:

```
{
  "state_id": "home_usuario_anonimo",
  "available_actions": ["doc_rag_17", "doc_rag_21", "doc_rag_44"],
  "action": "doc_rag_21",
  "action_probability": 0.18,
  "reward_terms": {
    "lectura_util": 1.0,
    "abandono": 0.0,
    "cost": -0.01
  }
}
```

Ese 0.18 es oro para ingeniería. Si luego comparamos otra política de recomendación, necesitamos saber cuánto se parecía o no se parecía a la política histórica que produjo el dato.

Agente con herramientas

El sistema decide si responder directamente, consultar RAG, llamar una tool, pedir confirmación o parar. El estado incluye tarea, permisos, resultados previos, presupuesto, trazas y contrato operativo.

CAMPO CRÍTICO	POR QUÉ IMPORTA
policy_version	La lógica de routing puede cambiar más que el modelo base.
reward_terms.cost	Una política puede resolver, pero gastar demasiado.
terminal	El agente puede cerrar la tarea o dejarla pendiente.
trace_id	Una decisión con spans, llamadas, coste y latencia.

Mini evento abreviado:

```
{
  "state_id": "tarea_con_datos_incompletos",
  "available_actions": ["consultar_rag", "llamar_tool", "pedir_confirmacion", "parar"],
  "action": "pedir_confirmacion",
  "action_probability": 0.27,
  "reward_terms": {
    "resolucion": 0.0,
    "cost": -0.05,
    "revision_evitable": 0.4,
    "reopened": 0.0
  }
}
```

Aquí se ve por qué `available_actions` importa tanto. Si `llamar_tool` no estaba permitida por permisos, la política no “falló” al no usarla. No podemos evaluar una acción que no era legal en ese estado.

Sistema educativo adaptativo

El sistema decide si dar una pista, mostrar una explicación, proponer un ejercicio más fácil o pasar al siguiente tema. El estado incluye dominio, intentos, tiempo, señales de confusión y objetivo pedagógico. La recompensa puede llegar tarde: quizá el alumno parece avanzar ahora, pero la comprensión real aparece en una prueba posterior.

CAMPO CRÍTICO	POR QUÉ IMPORTA
<code>reward_time</code>	La comprensión puede medirse después, no en el clic inmediato.
<code>state_features.intentos</code>	La misma pista no significa lo mismo en el primer intento que en el quinto.
<code>available_actions</code>	Algunas ayudas pueden no estar disponibles por nivel o secuencia didáctica.
<code>reward_terms</code>	Separar rapidez, comprensión y abandono evita optimizar solo velocidad.

Mini evento abreviado:

```
{
  "state_id": "ejercicio_gradientes_intento_2",
  "available_actions": ["dar_pista", "mostrar_solucion_parcial", "nuevo_ejercicio"],
  "action": "dar_pista",
  "action_probability": 0.41,
  "reward_time": "2026-06-08T08:00:00Z",
  "reward_terms": {
    "comprension_posterior": 0.8,
    "abandono": 0.0,
    "cost": -0.02
  }
}
```

Estos ejemplos no intentan cubrir todo RL. Intentan enseñar el patrón: si hay decisiones secuenciales, el dato debe conservar estado, acción, alternativas, probabilidad, consecuencia, tiempo y versión.

Cómo se vería en una base de datos

En producción, tarde o temprano alguien preguntará dónde viven estos datos. Un JSONL sirve para aprender y para pruebas pequeñas; un sistema de datos necesita tablas, claves, particionado, contratos y consultas.

Un modelo físico mínimo podría separar seis tablas:

TABLA	QUÉ GUARDA	PREGUNTA QUE RESPONDE
rl_events	Evento atómico de decisión.	Qué pasó en cada paso.
rl_episodes	Episodios agregados y retorno.	Cómo terminó cada trayectoria.
rl_policy_versions	Versiones de política y configuración.	Qué lógica produjo las acciones.
rl_reward_versions	Versiones de recompensa y ventana de atribución.	Qué objetivo estábamos optimizando.
rl_trajectory_snapshots	Cortes reproducibles del replay buffer.	Qué snapshot alimentó evaluación o entrenamiento.
rl_data_quality_runs	Resultados de gates y validaciones.	Qué se aceptó, revisó o bloqueó.

Ese esquema mínimo no pretende imponer una base de datos concreta. Pretende mostrar la anatomía profesional: eventos, episodios, versiones, snapshots y runs de calidad no deberían mezclarse en una tabla única sin gobierno.

También incluye `sql/query_examples.sql` con consultas que usarías en el día a día:

CONSULTA	PARA QUÉ SIRVE
Cobertura por <code>state_id</code> , <code>action</code> .	Detectar qué decisiones están respaldadas por datos.
Eventos sin propensión útil.	Encontrar filas que debilitan evaluación offline.
Recompensas tardías.	Auditar ventanas de atribución.
Episodios sin cierre terminal.	Detectar trayectorias incompletas.
Retorno por <code>policy_version</code> .	Comparar comportamiento histórico sin mezclar versiones.
Snapshots no publicables.	Impedir que un corte <code>review</code> o <code>block</code> alimente entrenamiento.

Esto conecta directamente con los facsímiles de operación y datos: un replay buffer serio no es solo un dataset; es un conjunto de tablas, contratos, validaciones, decisiones y evidencias.

Herramientas y SDKs que encajan aquí

Fecha de corte: 7 de junio de 2026. Esta lista no pretende decir “usa todas estas herramientas”. Pretende ubicar familias de herramientas en el pipeline. La regla importante es esta: ninguna herramienta arregla un evento mal diseñado. Si faltan `action_probability` , `policy_version` , `reward_time` o `available_actions` , el problema está en la instrumentación, no en el orquestador.

CAPA DEL PIPELINE	HERRAMIENTAS O SDKS	PARA QUÉ SIRVEN	QUÉ NO ARREGLAN
Validación de datos	Great Expectations, Pandera, Deequ	Reglas de schema, tipos, rangos, catálogos, nulos y expectativas.	No inventan propensión ni reconstruyen acciones disponibles.
Linaje y catálogo	OpenLineage, Marquez, DataHub	Registrar jobs, datasets, runs, propietarios, cambios y dependencias.	No sustituyen un contrato de evento RL.
Versionado y lakehouse	DVC, lakeFS, Delta Lake	Snapshots, rollback, control de cambios y reproducción de cortes.	No deciden si el reward está bien atribuido.
Orquestación	Airflow, Dagster, Prefect	Ejecutar ingesta, validación, backfills, snapshots y gates.	No convierte un pipeline sin checks en un pipeline fiable.
Feature store	Feast	Reutilizar señales online/offline y evitar inconsistencias de features.	No garantiza que una señal sea suficiente como estado de un MDP.
Observabilidad de datos	Evidently, WhyLabs/whylogs	Drift, calidad, distribución y alertas de datos.	No evalúan por sí solas una política nueva.
Tracking de experimentos	MLflow, Weights & Biases	Registrar runs, parámetros, métricas, artefactos, hashes y comparaciones.	No reemplazan el linaje del dataset ni el gate de datos.
RL y bandits	Vowpal Wabbit, Ray RLLib, TorchRL, CleanRL	Entrenar, simular o prototipar políticas cuando los datos ya son válidos.	No deberían ser el primer paso si el replay buffer aún no pasa contrato.
Warehouse o lakehouse	DuckDB, Postgres, BigQuery, Snowflake, Databricks	Consultas, agregados, auditorías y materialización de snapshots.	No corrigen eventos que llegaron incompletos.

Las herramientas de validación y contratos cubren la primera defensa del pipeline. Great Expectations, Pandera y Deequ documentan formas de expresar expectativas, schemas o tests sobre datos.¹¹¹²¹³ En este capítulo las usaremos para comprobar que `action` pertenece a `available_actions`, que `action_probability` está entre 0 y 1, que `reward_time` no precede a `event_time` y que todos los episodios tienen cierre.

Las herramientas de linaje y versionado ayudan a saber de dónde viene cada snapshot. OpenLineage, Marquez, DataHub, DVC, lakeFS y Delta Lake cubren piezas distintas de esa trazabilidad: eventos de linaje, visualización de metadatos, catálogo, versionado o

almacenamiento transaccional.¹⁴¹⁵¹⁶¹⁷¹⁸¹⁹ En un replay buffer, esto se traduce en una pregunta muy concreta: si alguien te da un modelo entrenado, ¿puedes recuperar exactamente qué eventos, contrato, versión de reward y política histórica lo alimentaron?

La orquestación y la observabilidad convierten el contrato en una rutina. Airflow, Dagster y Prefect sirven para programar y gobernar ejecuciones; Evidently y WhyLabs/whylogs para vigilar distribución, drift y calidad de datos; MLflow y Weights & Biases para registrar experimentos, métricas y artefactos.²⁰²¹²²²³²⁴²⁵²⁶ En un equipo real, este gate no viviría solo como comando manual: sería un paso de pipeline, con salida archivada y decisión visible.

Las librerías de RL llegan después. Vowpal Wabbit encaja especialmente bien cuando hablamos de contextual bandits y aprendizaje online; Ray RLlib cuando necesitas escala, simulación o entrenamiento distribuido; TorchRL cuando quieres trabajar dentro del ecosistema PyTorch; CleanRL cuando buscas implementaciones compactas para estudiar o prototipar algoritmos.²⁷²⁸²⁹³⁰ El orden importa: primero contrato e instrumentación; después validación y snapshots; luego evaluación offline; y solo entonces entrenamiento o despliegue de políticas.

También conviene pensar en SDKs propios. En un producto real, no quieres que cada equipo emita eventos RL “a mano”. Diseñarías un pequeño SDK interno con una función parecida a esta:

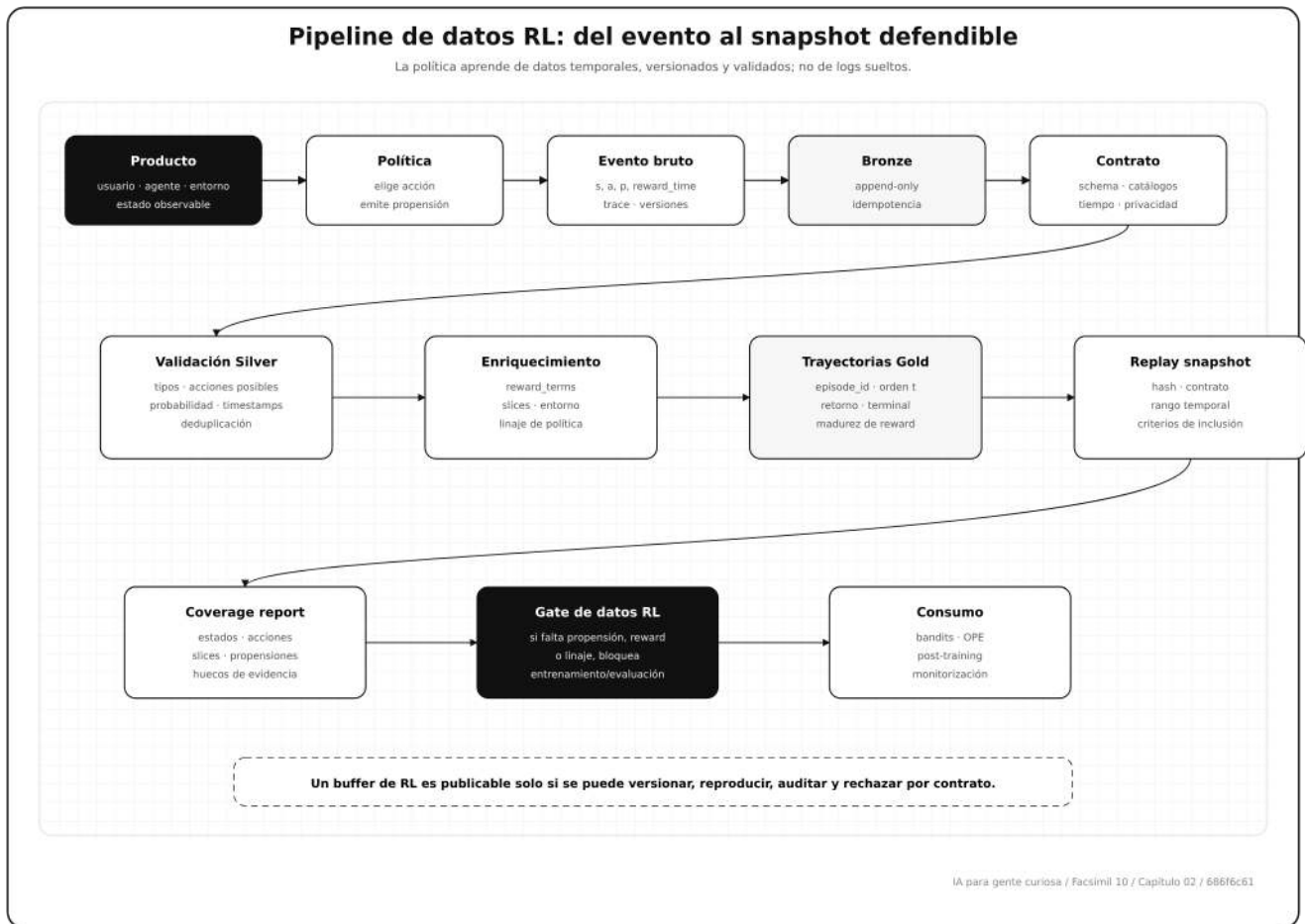
```

def emit_rl_event(
    episode_id,
    t,
    state,
    available_actions,
    action,
    action_probability,
    policy_version,
    reward=None,
    reward_version=None,
    trace_id=None,
):
    assert action in available_actions
    assert 0.0 <= action_probability <= 1.0
    return {
        "schema_version": "rl_event.v1",
        "episode_id": episode_id,
        "t": t,
        "state_id": state["state_id"],
        "state_features": state["features"],
        "available_actions": available_actions,
        "action": action,
        "action_probability": action_probability,
        "policy_version": policy_version,
        "reward": reward,
        "reward_version": reward_version,
        "trace_id": trace_id,
    }

```

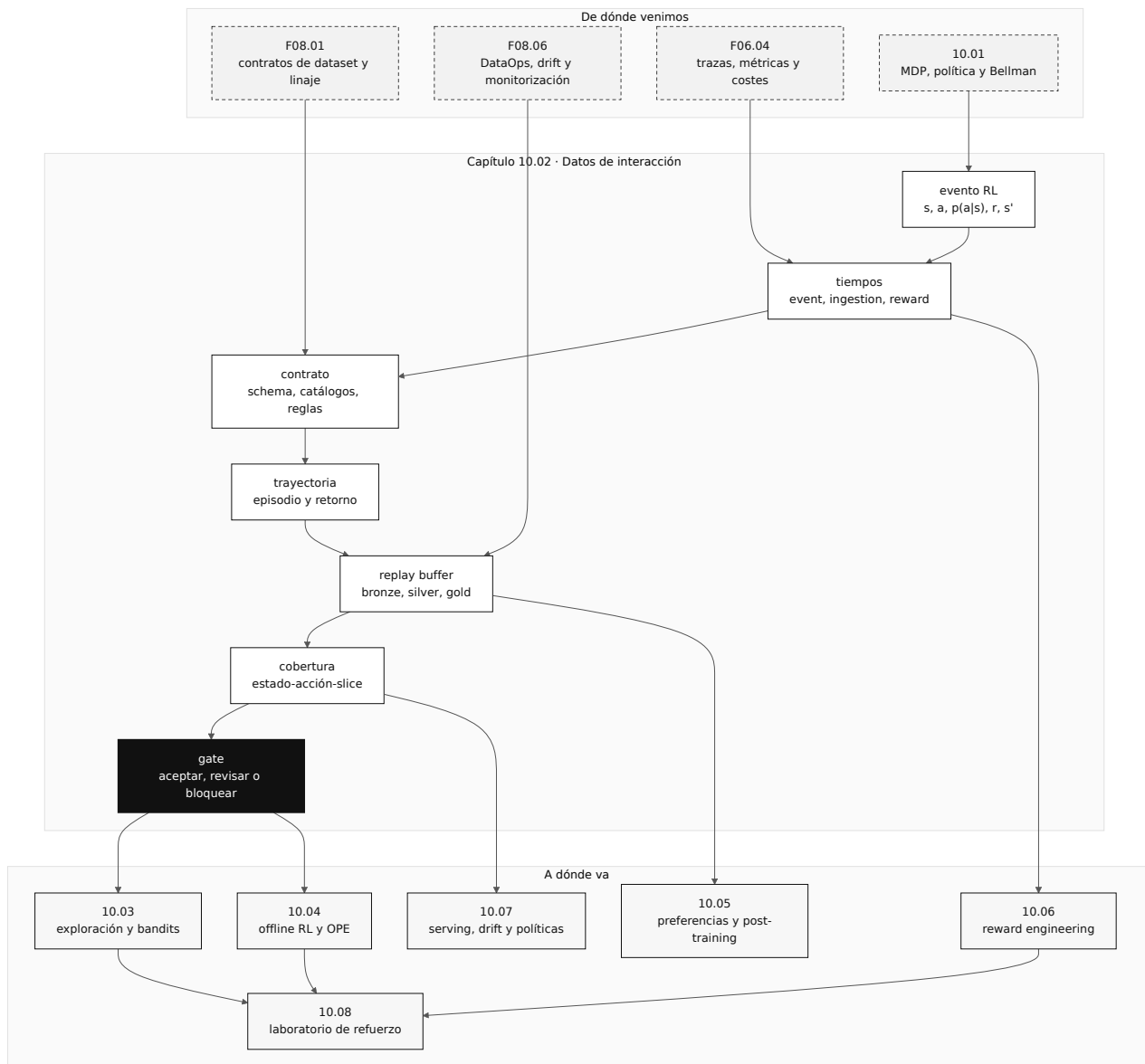
Ese SDK no es el pipeline completo. Es la primera línea de defensa: evita que producto emita eventos imposibles antes de que lleguen al lakehouse. Después vendrían validación de datos, linaje, snapshot, gate y tracking de runs.

Anatomía de un pipeline de datos para RL



Cómo encaja todo

Este mapa debe leerse en tres capas. Venimos del MDP del capítulo 01: estados, acciones, política y retorno. En este capítulo convertimos esos conceptos en datos versionados. Después, esos datos sostienen exploración, evaluación offline, post-training, reward engineering y operación de políticas.



Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Guardar logs sin política.	Parece suficiente para depurar.	Registrar <code>policy_id</code> , <code>policy_version</code> y <code>action_probability</code> .
Confundir evento con trayectoria.	Una fila se entiende mejor que una secuencia.	Agrupar por <code>episode_id</code> , ordenar por <code>t</code> y calcular retorno.
Calcular reward tarde y sin versión.	Es cómodo añadirlo después.	Guardar <code>reward_terms</code> , <code>reward_version</code> , <code>reward_time</code> y regla de atribución.
Olvidar acciones disponibles.	Miramos solo lo que ocurrió.	Registrar el catálogo permitido en cada estado.
Celebrar volumen sin cobertura.	Un agregado grande tranquiliza.	Medir cobertura por estado, acción, slice y política.
Tratar el replay buffer como carpeta.	Los experimentos empiezan rápido.	Publicar snapshots con contrato, hash, rango temporal y decisión.

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Evento de interacción	Registro atómico de estado, acción, probabilidad, recompensa y versión.
Trayectoria	Secuencia ordenada de eventos de un episodio.
Propensión	Probabilidad con la que la política histórica eligió la acción observada.
Recompensa tardía	Resultado que llega después de actuar y necesita regla de atribución.
Replay buffer	Almacén versionado de experiencias.
Snapshot de replay	Corte reproducible del buffer con contrato, hash y criterios de inclusión.
Cobertura estado-acción	Medida de qué parejas estado-acción están representadas en los datos.
Gate de datos RL	Validación que impide usar experiencias incompletas o no reproducibles.

Antes de pasar página

- ¿Puedes explicar por qué una trayectoria no es lo mismo que una fila?
- ¿Qué campo permite saber qué política produjo un evento?
- ¿Por qué `action_probability` importa para evaluación offline?
- ¿Qué diferencia hay entre `event_time`, `ingestion_time` y `reward_time`?
- ¿Qué pierde un replay buffer sin contrato ni hash?
- ¿Cómo medirías cobertura de una pareja estado-acción?
- ¿Qué harías si una acción crítica aparece con cobertura cero?
- ¿Qué artefactos produce el cuaderno del capítulo?

Para saber más

Apache Airflow. (2026). *Apache Airflow Documentation*.

<https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/>

AWS Labs. (2026). *Deequ: Unit Tests for Data*. <https://github.com/awslabs/deequ>

Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C. Y., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S. E., Wicke, M., Wilkiewicz, J., Zhang, X. y Zinkevich, M. (2017). TFX: A TensorFlow-based production-scale machine learning platform. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1387-1395.
<https://doi.org/10.1145/3097983.3098021>

Breck, E., Cai, S., Nielsen, E., Salib, M. y Sculley, D. (2017). The ML Test Score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data*, 1123-1132. <https://research.google/pubs/pub46555/>

CleanRL. (2026). *CleanRL Documentation*. <https://docs.cleanrl.dev/>

Dagster. (2026). *Dagster Documentation*. <https://docs.dagster.io/getting-started>

DataHub. (2026). *DataHub Documentation*. <https://docs.datahub.com/>

Delta Lake. (2026). *Delta Lake Documentation*. <https://docs.delta.io/>

Dudík, M., Langford, J. y Li, L. (2011). Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning*, 1097-1104.
https://icml.cc/2011/papers/511_icmlpaper.pdf

DVC. (2026). *What is DVC?* <https://dvc.org/doc/user-guide/what-is-dvc>

Evidently AI. (2026). *Data Drift Documentation*.
https://docs.evidentlyai.com/metrics/explainer_drift

Feast. (2026). *Feast Documentation*. <https://docs.feast.dev/>

Great Expectations. (2026). *Expectations Overview*.
https://docs.greatexpectations.io/docs/cloud/expectations/expectations_overview/

lakeFS. (2026). *lakeFS Documentation*. <https://docs.lakefs.io/>

Li, L., Chu, W., Langford, J. y Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th International Conference on World Wide Web*, 661-670. <https://doi.org/10.1145/1772690.1772758>

Marquez Project. (2026). *Marquez*. <https://github.com/MarquezProject/marquez>

MLflow. (2026). *MLflow Tracking*. <https://mlflow.org/docs/latest/ml/tracking>

OpenLineage. (2026). *OpenLineage Documentation*. <https://openlineage.io/docs/>

Pandera. (2026). *Pandera documentation*. <https://pandera.readthedocs.io/en/stable/>

Prefect. (2026). *Deployments*. <https://docs.prefect.io/concepts/deployments>

PyTorch. (2026). *TorchRL Documentation*. <https://docs.pytorch.org/rl/>

Ray. (2026). *RLlib: Industry-Grade, Scalable Reinforcement Learning*.
<https://docs.ray.io/en/latest/rllib/>

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F. y Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 28.
https://papers.nips.cc/paper_files/paper/2015/hash/86df7dcfd896fcdf2674f757a2463eba-Abstract.html

Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.^a ed.). MIT Press. <https://incompleteideas.net/book/the-book-2nd.html>

TensorFlow. (2026). *ML Metadata*. <https://tensorflow.github.io/tfx/guide/mlmd/>

Vowpal Wabbit. (2026). *Vowpal Wabbit Documentation*. <https://vowpalwabbit.org/docs/>

Weights & Biases. (2026). *Experiments Overview*. <https://docs.wandb.ai/models/track>

WhyLabs. (2026). *WhyLabs Documentation*. <https://docs.whylabs.ai/docs/>

En resumen

IDEA	QUÉ DEBES RECORDAR
RL necesita datos temporales.	El orden estado-acción-recompensa importa tanto como los valores.
Un evento debe registrar alternativas.	Sin <code>available_actions</code> , no sabemos qué podía decidir la política.
La propensión no es un lujo.	Sin probabilidad histórica, muchas evaluaciones offline pierden rigor.
La recompensa puede llegar tarde.	Necesitas <code>reward_time</code> y regla de atribución.
El replay buffer es un producto de datos.	Debe tener contrato, linaje, cobertura, privacidad, versión y hash.
El capítulo 03 depende de este.	Explorar con bandits exige logging y propensiones bien capturadas.

Notas

1. Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.^a ed.). MIT Press. <https://incompleteideas.net/book/the-book-2nd.html> ↵
2. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F. y Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 28. https://papers.nips.cc/paper_files/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html ↵
3. Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C. Y., Lew, L., Mewald, C., Modi, A. N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S. E., Wicke, M., Wilkiewicz, J., Zhang, X. y Zinkevich, M. (2017). TFX: A TensorFlow-based production-scale machine learning platform. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1387-1395. <https://doi.org/10.1145/3097983.3098021> ↵
4. Breck, E., Cai, S., Nielsen, E., Salib, M. y Sculley, D. (2017). The ML Test Score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data*, 1123-1132. <https://research.google/pubs/pub46555/> ↵
5. TensorFlow. (2026). *ML Metadata*. <https://tensorflow.github.io/tfx/guide/mlmd/>. Consultado el 6 de junio de 2026. ↵

6. OpenLineage. (2026). *OpenLineage Documentation*. <https://openlineage.io/docs/>. Consultado el 6 de junio de 2026. ↵
7. Dudík, M., Langford, J. y Li, L. (2011). Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning*, 1097-1104. https://icml.cc/2011/papers/511_icmlpaper.pdf ↵
8. Li, L., Chu, W., Langford, J. y Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th International Conference on World Wide Web*, 661-670. <https://doi.org/10.1145/1772690.1772758> ↵
9. Great Expectations. (2026). *Expectations Overview*. https://docs.greatexpectations.io/docs/cloud/expectations/expectations_overview/. Consultado el 6 de junio de 2026. ↵
10. Pandera. (2026). *Pandera documentation*. <https://pandera.readthedocs.io/en/stable/>. Consultado el 6 de junio de 2026. ↵
11. Great Expectations. (2026). *Expectations Overview*. https://docs.greatexpectations.io/docs/cloud/expectations/expectations_overview/. Consultado el 6 de junio de 2026. ↵
12. Pandera. (2026). *Pandera documentation*. <https://pandera.readthedocs.io/en/stable/>. Consultado el 6 de junio de 2026. ↵
13. AWS Labs. (2026). *Deequ: Unit Tests for Data*. <https://github.com/awslabs/deequ>. Consultado el 6 de junio de 2026. ↵
14. OpenLineage. (2026). *OpenLineage Documentation*. <https://openlineage.io/docs/>. Consultado el 6 de junio de 2026. ↵
15. Marquez Project. (2026). *Marquez*. <https://github.com/MarquezProject/marquez>. Consultado el 7 de junio de 2026. ↵
16. DataHub. (2026). *DataHub Documentation*. <https://docs.datahub.com/>. Consultado el 6 de junio de 2026. ↵
17. DVC. (2026). *What is DVC?*. <https://dvc.org/doc/user-guide/what-is-dvc>. Consultado el 6 de junio de 2026. ↵
18. lakeFS. (2026). *lakeFS Documentation*. <https://docs.lakefs.io/>. Consultado el 6 de junio de 2026. ↵
19. Delta Lake. (2026). *Delta Lake Documentation*. <https://docs.delta.io/>. Consultado el 6 de junio de 2026. ↵
20. Apache Airflow. (2026). *Apache Airflow Documentation*. <https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/>. Consultado el 7 de junio de 2026. ↵

- !1. Dagster. (2026). *Dagster Documentation*. <https://docs.dagster.io/getting-started>. Consultado el 7 de junio de 2026. ↵
- !2. Prefect. (2026). *Deployments*. <https://docs.prefect.io/concepts/deployments>. Consultado el 7 de junio de 2026. ↵
- !3. Evidently AI. (2026). *Data Drift Documentation*. https://docs.evidentlyai.com/metrics/explainer_drift. Consultado el 6 de junio de 2026. ↵
- !4. WhyLabs. (2026). *WhyLabs Documentation*. <https://docs.whylabs.ai/docs/>. Consultado el 7 de junio de 2026. ↵
- !5. MLflow. (2026). *MLflow Tracking*. <https://mlflow.org/docs/latest/ml/tracking>. Consultado el 7 de junio de 2026. ↵
- !6. Weights & Biases. (2026). *Experiments Overview*. <https://docs.wandb.ai/models/track>. Consultado el 7 de junio de 2026. ↵
- !7. Vowpal Wabbit. (2026). *Vowpal Wabbit Documentation*. <https://vowpalwabbit.org/docs/>. Consultado el 7 de junio de 2026. ↵
- !8. Ray. (2026). *RLlib: Industry-Grade, Scalable Reinforcement Learning*. <https://docs.ray.io/en/latest/rllib/>. Consultado el 7 de junio de 2026. ↵
- !9. PyTorch. (2026). *TorchRL Documentation*. <https://docs.pytorch.org/rl/>. Consultado el 7 de junio de 2026. ↵
- !0. CleanRL. (2026). *CleanRL Documentation*. <https://docs.cleanrl.dev/>. Consultado el 7 de junio de 2026. ↵

Capítulo 03: Exploración, bandits y validación de políticas

Aprender mientras decides no es gratis

En un A/B test clásico repartes tráfico, esperas, analizas y decides. En un bandit, el sistema decide y aprende a la vez: manda más tráfico a lo que parece funcionar, pero reserva parte para seguir probando. Suena eficiente. También es delicado, porque la exploración deja de ser un análisis externo y pasa a modificar la experiencia real del producto.

La pregunta de ingeniería no es “¿usamos bandits?”. La pregunta es:

¿qué podemos probar, con qué límite, en qué casos, con qué métrica de daño aceptable y con qué rollback?

Un bandit es una versión simplificada de aprendizaje por refuerzo. No modela una secuencia larga como un MDP completo. En cada ronda observas un contexto opcional, eliges una acción, recibes una recompensa y actualizas evidencia. Sutton y Barto tratan los bandits como el caso mínimo donde aparece el dilema entre explorar y explotar.¹

Qué no es un bandit

Un bandit no es “un A/B test que se mueve solo”. Tampoco es una forma elegante de saltarse evaluación. Y no es una excusa para probar cualquier opción con cualquier usuario.

MALENTENDIDO	POR QUÉ CONFUNDE	FORMA CORRECTA DE PENSARLO
“El bandit encontrará solo lo mejor”.	Si la recompensa está mal diseñada, optimiza lo equivocado.	Primero reward, límites y trazas; luego política.
“Explorar siempre mejora”.	Explorar opciones peores tiene coste real.	Medir regret y presupuesto de exploración.
“Si cambia asignación, ya aprende”.	Cambiar tráfico no garantiza evidencia útil.	Registrar acción, propensión, contexto y recompensa.
“Es más moderno que A/B testing”.	Responde otra pregunta.	Bandit optimiza durante la asignación; A/B estima con diseño fijo.
“Vale para cualquier caso”.	Algunos contextos no toleran exploración automática.	Definir slices donde se limita o desactiva.

En ingeniería de IA, un bandit puede ser útil para routing de modelos, selección de prompts, configuración de RAG, elección de herramientas o recomendación. Pero solo si aceptamos la frase incómoda: aprender tiene coste de oportunidad.

A/B test, bandit o ninguna de las dos

Esta distinción importa mucho en proyectos reales. Si el objetivo es estimar con claridad el efecto de una variante, un A/B test suele ser más limpio: asignación fija, periodo definido, análisis estadístico y decisión. Si el objetivo es repartir tráfico mientras aprendemos y el coste de equivocarnos está acotado, un bandit puede tener sentido. Si la decisión afecta a casos críticos, a cumplimiento normativo o a una experiencia donde el error tiene consecuencias serias, quizá no corresponde explorar online: primero simulación, evaluación offline y revisión humana del cambio.

SITUACIÓN	MEJOR PUNTO DE PARTIDA	POR QUÉ
Quiero saber si un nuevo prompt mejora una métrica antes de adoptarlo.	A/B test controlado.	Separas estimación de despliegue y puedes medir efecto con menos movimiento adaptativo.
Tengo varias rutas de modelo y el coste de una mala elección está limitado.	Bandit con gates.	El sistema puede mover tráfico hacia rutas prometedoras mientras controla regret, coste y slices.
El reward llega tarde o depende de varias decisiones encadenadas.	MDP, offline RL o evaluación de trayectorias.	Un bandit ve una recompensa inmediata o atribuible a una decisión; no modela bien una secuencia larga.
La métrica principal no está validada.	Ninguna política adaptativa todavía.	Si el reward está mal, el algoritmo aprenderá a optimizar ruido o un proxy cómodo.
Hay poco volumen.	A/B simple, regla fija o simulación.	Un bandit necesita observaciones suficientes para que la incertidumbre se reduzca.
El cambio solo debe activarse para un grupo por despliegue progresivo.	Feature flag o rollout gradual.	Controlas exposición, pero no necesariamente aprendes una política.

Una forma práctica de decidirlo es preguntar: ¿necesito aprender una asignación durante el uso, o solo necesito controlar quién ve una versión? La primera pregunta apunta a bandits. La segunda apunta a feature flags, canary releases o experimentación clásica. LaunchDarkly documenta rollouts porcentuales para liberar cambios de forma gradual, y OpenFeature separa la evaluación de flags mediante un contexto que puede incluir atributos de usuario, aplicación o petición.²³

El problema formal

En un bandit de K brazos tenemos un conjunto de acciones:

$$A = \{1, 2, \dots, K\}$$

En cada ronda t , elegimos una acción a_t y observamos una recompensa:

$$r_t \sim R(a_t)$$

SÍMBOL O	SIGNIFICADO	EJEMPLO
A	Conjunto de acciones o brazos.	<code>modelo_rapido</code> , <code>modelo_fuerte</code> , <code>revision_humana</code> .
K	Número de acciones.	3
t	Ronda de decisión.	Ticket número 41 de la ventana.
a_t	Acción elegida en la ronda t .	Usar <code>modelo_fuerte</code> .
r_t	Recompensa observada.	Calidad menos coste y penalización por reapertura.
$R(a_t)$	Distribución de recompensas de esa acción.	El modelo fuerte suele dar más calidad, pero cuesta más.

La media real de una acción, que normalmente no conocemos, es:

$$\mu_a = \mathbb{E}[r \mid a]$$

Y la media observada después de probarla $N_a(t)$ veces es:

$$\hat{\mu}_{a,t} = \frac{1}{N_a(t)} \sum_{i:a_i=a} r_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
μ_a	Recompensa media real de la acción.	Valor esperado del modelo fuerte.
$\hat{\mu}_{a,t}$	Media estimada con datos hasta t .	Media observada tras 12 usos.
$N_a(t)$	Veces que se ha elegido a .	12
$\sum_{i:a_i=a} r_i$	Suma de recompensas donde se eligió a .	8,9 puntos acumulados.

La trampa está aquí: una acción con poca muestra puede parecer mala por azar o buena por azar. La exploración es la disciplina de no confundir evidencia temprana con verdad.

Greedy y epsilon-greedy

La política greedy elige siempre la acción con mejor media observada:

$$a_t = \arg \max_a \hat{\mu}_{a,t}$$

Es simple, rápida y fácil de explicar. También puede quedar atrapada. Si una acción buena empezó con dos recompensas flojas, greedy quizá no vuelva a probarla nunca.

ϵ -greedy añade exploración:

$$a_t = \begin{cases} \text{acción aleatoria,} & \text{con probabilidad } \epsilon \\ \arg \max_a \hat{\mu}_{a,t}, & \text{con probabilidad } 1 - \epsilon \end{cases}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
ϵ	Probabilidad de explorar.	0,1
$1 - \epsilon$	Probabilidad de explotar la mejor media observada.	0,9
$\arg \max_a$	Acción con mayor estimación.	modelo_fuerte .

ϵ -greedy es buena para aprender el concepto porque es auditable. Puedes decir “exploramos un 10 %”. Pero tiene una limitación clara: explora de forma poco informada. Puede seguir probando acciones con mala evidencia aunque haya otras con incertidumbre más razonable.

UCB: explorar donde falta evidencia

UCB, *Upper Confidence Bound*, elige la acción con mejor combinación de media observada e incertidumbre:

$$a_t = \arg \max_a \left(\hat{\mu}_{a,t} + c \frac{\ln t}{N_a(t)} \right)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\hat{\mu}_{a,t}$	Media observada de la acción.	0,74
c	Peso del bonus de exploración.	0,8
$\ln t$	Logaritmo de la ronda actual.	$\ln 100$
$N_a(t)$	Veces que se ha probado la acción.	5
$c \frac{\ln t}{N_a(t)}$	Bonus por incertidumbre.	Alto si hay poca muestra.

La intuición es bonita: no explora por capricho, explora donde la incertidumbre todavía es grande. Auer, Cesa-Bianchi y Fischer analizaron garantías finitas para el problema multi-armed bandit, incluyendo UCB1.⁴ Antes, Lai y Robbins habían establecido fundamentos asintóticos sobre reglas adaptativas eficientes.⁵

Un ejemplo pequeño ayuda. Imagina dos rutas de modelo en la ronda $t = 100$. La ruta A tiene media observada 0,72 tras 40 usos. La ruta B tiene media observada 0,68 tras 5 usos. Si usamos $c = 0,4$, UCB calcula:

$$\text{score}(A) = 0,72 + 0,4 \frac{\ln 100}{40} \approx 0,856$$

$$\text{score}(B) = 0,68 + 0,4 \frac{\ln 100}{5} \approx 1,064$$

El score no es una probabilidad; puede superar 1. Es una puntuación de decisión que mezcla media e incertidumbre. Aunque B tiene peor media observada, UCB la prueba porque apenas tiene muestra. Si después de varias rondas B sigue sin mejorar, su bonus baja porque $N_B(t)$ crece. Esto es lo que queremos en producción: no “probar por probar”, sino probar donde la evidencia todavía no permite cerrar el caso.

RUT A	MEDIA OBSERVADA	USO S	BONUS UCB	SCOR E	LECTURA
A	0,72	40	0,136	0,856	Buena evidencia, menor incertidumbre.
B	0,68	5	0,384	1,064	Peor media inicial, pero mucha incertidumbre.

Thompson sampling: decidir según creencias

Thompson sampling viene de una idea de 1933: representar la incertidumbre sobre cada opción y muestrear de esa creencia antes de decidir.⁶

En el caso Bernoulli, donde cada acción produce éxito o fracaso, podemos usar una distribución Beta:

$$\theta_a \sim \text{Beta}(\alpha_a, \beta_a)$$

Elegimos:

$$a_t = \arg \max_a \theta_a$$

Después actualizamos:

$$(\alpha_a, \beta_a) = \begin{cases} (\alpha_a + 1, \beta_a), & \text{si hay éxito} \\ (\alpha_a, \beta_a + 1), & \text{si hay fracaso} \end{cases}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
θ_a	Muestra de la creencia de éxito para la acción a .	0,73
α_a	Evidencia acumulada de éxitos.	8
β_a	Evidencia acumulada de fracasos.	3
Beta	Distribución sobre probabilidades entre 0 y 1.	Creencia sobre tasa de éxito.

Chapelle y Li evaluaron empíricamente Thompson sampling y mostraron su buen comportamiento práctico en problemas de bandits.⁷ Para un equipo de ingeniería, su ventaja es que explora de forma proporcional a la incertidumbre. Su coste pedagógico es que puede ser más difícil de explicar que UCB a personas que no viven en estadística bayesiana.

Regret: la factura de aprender

Si en cada ronda existe una mejor acción disponible con media μ^* , el regret acumulado se escribe:

$$\text{Regret}(T) = \sum_{t=1}^T (\mu^* - \mu_{a_t})$$

En un contexto con recompensas conocidas de simulación, también podemos calcular regret por ronda como:

$$\text{regret}_t = r_t^* - r_t$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\text{Regret}(T)$	Pérdida acumulada por no elegir siempre lo mejor.	3,8
T	Número de rondas.	60
μ^*	Media de la mejor acción.	0,78
μ_{a_t}	Media de la acción elegida.	0,61
r_t^*	Mejor recompensa disponible en la ronda simulada.	0,82
r_t	Recompensa recibida por la acción elegida.	0,57

Bubeck y Cesa-Bianchi revisan el análisis de regret en bandits estocásticos y no estocásticos.⁸ La traducción práctica: no basta con mirar recompensa acumulada. Hay que mirar cuánto coste asumimos por aprender.

Cuando las recompensas cambian

Hasta aquí hemos hablado como si la distribución de recompensa de cada acción fuera relativamente estable. En productos reales casi nunca es tan cómodo. El modelo rápido puede recibir una mejora de proveedor. El modelo fuerte puede subir de precio. El tráfico puede cambiar por calendario académico. Un prompt puede dejar de funcionar porque el tipo de consulta cambia. Esto se llama no estacionariedad: la recompensa que estimabas ayer no describe necesariamente la recompensa de hoy.

Una solución simple es estimar medias con ventana móvil: solo cuentan las últimas W rondas.

$$\hat{\mu}_{a,t}^{(W)} = \frac{1}{N_{a,W}(t)} \sum_{i: a_i=a, t-W < i \leq t} r_i$$

Otra solución es aplicar decaimiento: las observaciones recientes pesan más que las antiguas.

$$\hat{\mu}_{a,t}^{(\lambda)} = \frac{\sum_{i: a_i=a} \lambda^{t-i} r_i}{\sum_{i: a_i=a} \lambda^{t-i}}, \quad 0 < \lambda \leq 1$$

SÍMBOL O	SIGNIFICADO	DECISIÓN DE INGENIERÍA
W	Tamaño de la ventana móvil.	Ventanas cortas reaccionan rápido, pero son más ruidosas.
$N_{a,W}(t)$	Veces que la acción a aparece dentro de la ventana.	Si es bajo, no publiques conclusión fuerte.
λ	Factor de decaimiento temporal.	Cerca de 1 conserva historia; más bajo olvida antes.
λ^{t-i}	Peso de una observación antigua.	Cuanto más antigua, menos pesa.

En una plataforma de IA esto se traduce en tres prácticas: versionar cada modelo o prompt como acción distinta, reiniciar o separar estadísticas cuando cambian condiciones importantes, y monitorizar reward drift por slice. Si mezclas observaciones de `modelo_fuerte_v1` y `modelo_fuerte_v2`, quizá el algoritmo parezca estable, pero en realidad has borrado el linaje de la decisión.

Contextual bandits: cuando el estado importa, pero no hay trayectoria larga

Un contextual bandit observa contexto x_t , elige acción a_t y recibe recompensa r_t :

$$x_t \rightarrow a_t \rightarrow r_t$$

La política se escribe:

$$\pi(a \mid x) = P(A_t = a \mid X_t = x)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x_t	Contexto observado antes de decidir.	Idioma, criticidad, coste, tipo de consulta.
a_t	Acción elegida.	Prompt A, modelo fuerte, top-k 6.
r_t	Recompensa observada.	Eval aprobada menos coste.
$\pi(a \mid x)$	Probabilidad de elegir acción dado el contexto.	0,22 para <code>modelo_fuerte</code> .

La diferencia con un MDP completo es que no modelamos una secuencia larga de estados. No decimos que la acción no tenga consecuencias futuras; decimos que para este problema vamos a optimizar una decisión repetida con contexto inmediato. Li y coautores aplicaron contextual bandits a recomendación personalizada de noticias, registrando acciones y recompensas para aprender asignaciones adaptativas.⁹

Ejemplos que sí se parecen a proyectos de IA

Routing de modelos

Tienes tres rutas: modelo rápido, modelo fuerte y revisión humana. La recompensa no debería ser solo “respuesta aceptada”. Debería descontar coste, latencia, reapertura y fallos de groundedness.

ACCIÓN	SEÑAL POSITIVA	COSTE QUE DEBES RESTAR
modelo_rapido	Baja latencia y coste.	Puede fallar en casos complejos.
modelo_fuerte	Más calidad en casos difíciles.	Más coste y latencia.
revision_humana	Alta fiabilidad en casos críticos.	Capacidad limitada y demora.

Un bandit aquí no decide “qué modelo es mejor en abstracto”. Decide cómo repartir tráfico bajo restricciones.

Selección de prompts

Puedes probar prompt A, B o C para una tarea estable. La recompensa puede venir de una eval automática más coste de tokens:

$$r = \mathbf{1}\{\text{eval pasa}\} - \lambda \cdot \text{tokens}$$

SÍMBOLO	SIGNIFICADO
$\mathbf{1}\{\text{eval pasa}\}$	1 si la salida supera la evaluación, 0 si no.
λ	Penalización por token.
tokens	Tokens usados por la respuesta.

Si no restas tokens, el sistema puede aprender que el prompt más largo “gana” aunque sea más caro sin aportar valor.

Configuración de RAG

Puedes elegir `top_k=3` , `top_k=6` , búsqueda híbrida o reranking. La recompensa puede combinar cita correcta, groundedness y latencia.

ACCIÓN	QUÉ APRENDE EL BANDIT	RIESGO SI NO LIMITAS
<code>top_k_3</code>	Menos contexto, menor latencia.	Puede perder evidencia.
<code>top_k_6</code>	Más cobertura documental.	Más coste y ruido.
<code>hybrid_search</code>	Mejor recall en consultas raras.	Más complejidad.
<code>rerank</code>	Mejor orden final.	Más latencia.

Este ejemplo conecta con el facsímil 4: una política de RAG no se evalúa solo por respuesta final, sino por recuperación, cita, coste y trazas.

Validar una política antes de ponerla a decidir

Una política bandit en producción necesita gates. No basta con decir que UCB obtuvo más recompensa en simulación. Tenemos que fijar límites:

LÍMITE	PREGUNTA DE INGENIERÍA	ARTEFACTO
Presupuesto de exploración	¿Qué porcentaje máximo puede probar opciones inciertas?	<code>max_exploration_share</code> .
Regret máximo	¿Cuánto coste de aprendizaje aceptamos por ventana?	<code>max_regret</code> .
Coste máximo	¿Cuánto puede gastar la política por lote?	SLO de coste.
Calidad mínima	¿Qué opción se retira si cae por debajo del umbral?	gate de evaluación.
Slices sin exploración	¿Dónde no permitimos exploración automática?	política por criticidad.
Propensión registrada	¿Podremos evaluar offline después?	<code>action_probability</code> en cada evento.
Rollback	¿Cómo volvemos a una política fija?	feature flag y runbook.

Podemos escribir un gate simple:

$$\text{publicable} = (\text{Regret}(T) \leq R_{\max}) \wedge (B_T \leq B_{\max}) \wedge (C_T \leq C_{\max}) \wedge (Q_T \geq Q_{\min})$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$R_{\max}$	Regret máximo permitido.	4,0
B_T	Fracción de rondas exploratorias.	0,18
$B_{\max}$	Presupuesto máximo de exploración.	0,25
C_T	Coste acumulado o medio.	8,4
$C_{\max}$	Coste máximo permitido.	10
Q_T	Calidad mínima medida en la ventana.	0,82
$Q_{\min}$	Umbral mínimo de calidad.	0,75

Dudík, Langford y Li explican evaluación y aprendizaje de políticas con estimadores que combinan propensión y modelos, lo que refuerza la necesidad de registrar probabilidades de acción desde el principio.¹⁰

El evento mínimo de una ronda bandit

El capítulo 02 insistía en eventos, trayectorias y linaje. Aquí el evento se vuelve todavía más importante, porque una política adaptativa cambia la probabilidad de observar ciertas acciones. Si no registras esa probabilidad, después no puedes reconstruir bien qué habría pasado con otra política.

Un evento útil no se limita a `action` y `reward`. Necesita contexto, acciones permitidas, política, probabilidad de asignación, motivo de elección y versión del contrato:

```
{
  "event_type": "bandit_round",
  "event_id": "evt_20260608_000184",
  "trace_id": "trc_support_0194",
  "occurred_at": "2026-06-08T10:12:44Z",
  "policy_version": "routing_bandit.v3",
  "contract_version": "bandit_policy_gate.v1",
  "context": {
    "slice": "media_criticidad",
    "language": "es",
    "estimated_complexity": 0.74,
    "customer_tier": "standard"
  },
  "allowed_actions": ["modelo_rapido", "modelo_fuerte", "revision_humana"],
  "selected_action": "modelo_fuerte",
  "action_probability": 0.37,
  "selection_reason": "ucb_score",
  "reward": {
    "quality": 0.86,
    "latency_ms": 1840,
    "cost_eur": 0.021,
    "reopened": false,
    "final_reward": 0.742
  }
}
```

Ese JSON no es decoración. Es el expediente técnico de la decisión. `allowed_actions` evita evaluar una política contra acciones que no estaban disponibles. `action_probability` permite estimadores offline. `selection_reason` distingue exploración inicial, UCB, Thompson o política estable. `contract_version` permite saber qué límites estaban vigentes. Y `trace_id` conecta la ronda con el sistema completo: prompt, modelo, retrieval, herramientas y métricas.

De simulación a piloto controlado

El orden sano no es “simulación buena, producción”. El orden sano es una escalera:

FASE	QUÉ OCURRE	QUÉ DEBE SALIR
Replay histórico	Reproduces decisiones sobre eventos antiguos con recompensas conocidas o estimadas.	Scorecard offline y errores de cobertura.
Simulación sintética	Creas escenarios pequeños donde sabes qué acción debería ganar.	Prueba de que el código del algoritmo no hace cosas absurdas.
Shadow	La política recomienda, pero no decide. El sistema ejecuta la política estable.	Comparación entre recomendación y acción real, sin afectar usuarios.
Piloto limitado	La política decide solo en slices permitidos y con tráfico bajo.	Métricas online, regret, coste, calidad y rollback probado.
Producción gradual	Aumentas exposición por ventanas y gates.	Evidencia acumulada y alertas por drift.

En un piloto controlado debes escribir antes de lanzar:

ELEMENTO	REGLA CONCRETA
Feature flag	La política bandit vive detrás de una flag con variante <code>stable</code> y variante <code>bandit_candidate</code> .
Política de reserva	Si falta contexto, reward o trazabilidad, se usa la ruta estable.
Slices permitidos	Solo baja o media criticidad hasta que el comité técnico apruebe ampliar.
Límite de exposición	Por ejemplo, 5 % durante 24 horas, 10 % durante 48 horas y revisión antes de subir.
Condiciones de parada	Regret por encima del umbral, coste medio alto, caída de calidad, falta de trazas o desviación de tráfico.
Observabilidad	Dashboard por política, acción, slice, coste, reward, latencia y drift.

OpenTelemetry mantiene convenciones semánticas para atributos de feature flags, de modo que decisiones de flags pueden aparecer en trazas y métricas con nombres estables.¹¹ Esto encaja bien con bandits: la decisión adaptativa no debe vivir en una caja aislada, sino en la misma trazabilidad que el resto del sistema.

Herramientas y SDKs

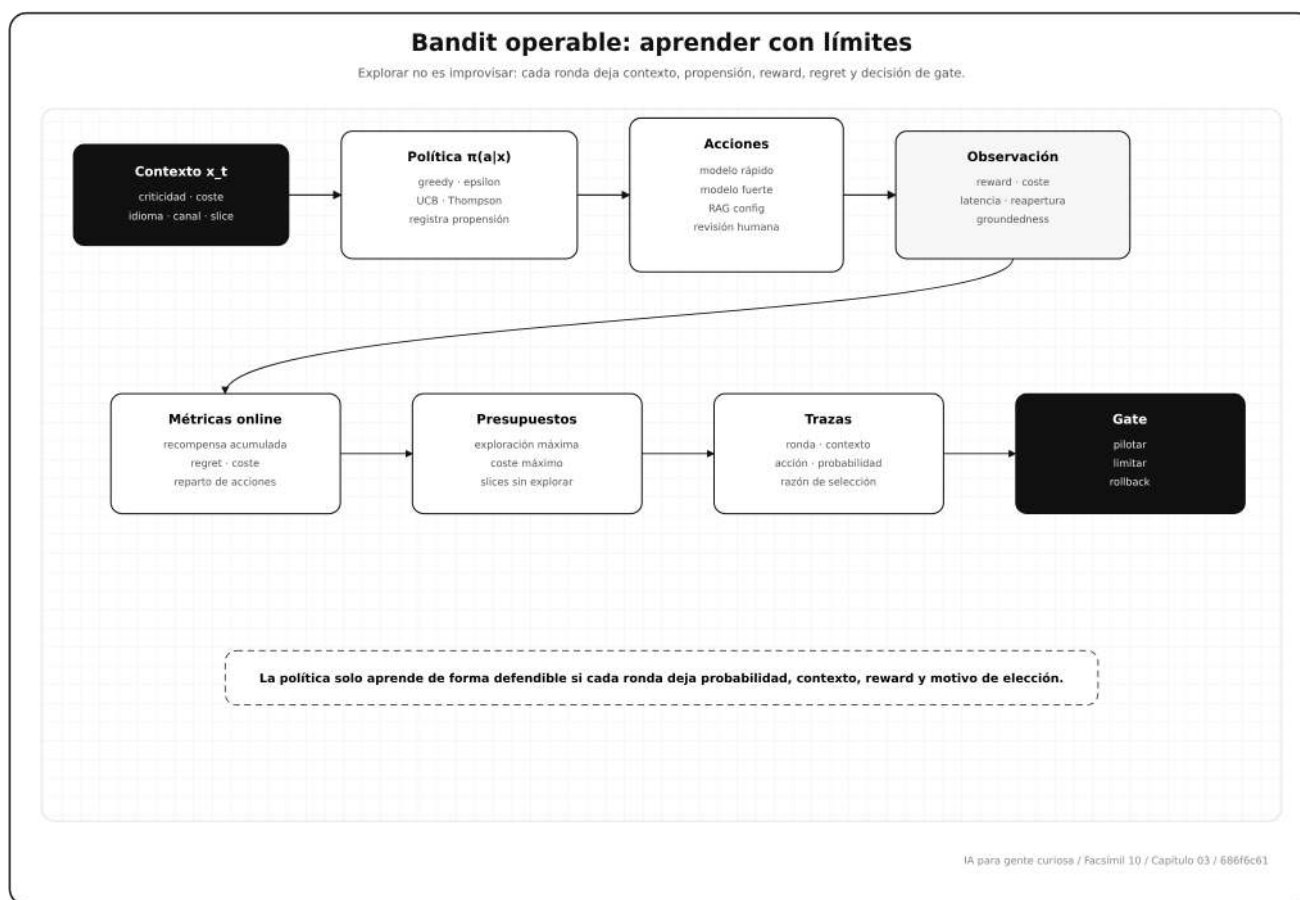
Fecha de corte: 8 de junio de 2026. En este capítulo las herramientas tienen que entenderse como soporte, no como sustituto del diseño estadístico.

HERRAMIENTA	ENCAJA CUANDO	QUÉ MIRAR ANTES
Vowpal Wabbit	Contextual bandits, aprendizaje online y experimentación con políticas.	Logging de propensión, formato de ejemplos y evaluación.
Ray RLlib	Simulación o entrenamiento RL a escala.	Si el problema realmente exige RL completo o basta bandit.
TorchRL	Prototipos dentro del ecosistema PyTorch.	Entorno, tensores, collectors y reproducibilidad.
CleanRL	Estudiar algoritmos con implementaciones compactas.	No confundir ejemplo educativo con plataforma productiva.
OpenFeature	API neutral de proveedor para evaluar flags con contexto.	Separar decisión de flag, decisión bandit y trazabilidad.
LaunchDarkly	Rollouts, flags, targeting y experimentación de producto.	Variantes, off variation, rollout gradual y deuda de flags.
Unleash	Feature flags con estrategias de activación y gradual rollout.	Contexto de evaluación, stickiness, segmentos y operación self-hosted si aplica.
Statsig	Feature gates, experimentos y análisis de producto.	Diferencia entre gate, experimento, métricas y assignment.
GrowthBook	Experimentación y feature flags conectadas con analítica.	Si el equipo quiere evaluación apoyada en warehouse y SDKs.
Eppo	Rollouts, interruptores de parada y análisis de experimentos.	Métricas en warehouse, asignación y criterios de lectura.
Optimizely Feature Experimentation	A/B tests y feature flags en productos con cultura de experimentación.	Diferenciar experimento, rollout y personalización.
Langfuse o Phoenix	Trazas y evaluación de aplicaciones LLM.	Relacionar decisión bandit con prompt, retrieval, coste y evaluación.

Vowpal Wabbit documenta soporte para aprendizaje online y contextual bandits; RLlib, TorchRL y CleanRL cubren distintos niveles de entrenamiento, investigación y prototipado RL.^{[12131415](#)} Para nuestro caso, lo importante es el orden: antes de usar una librería, valida evento, propensión, reward y gate.

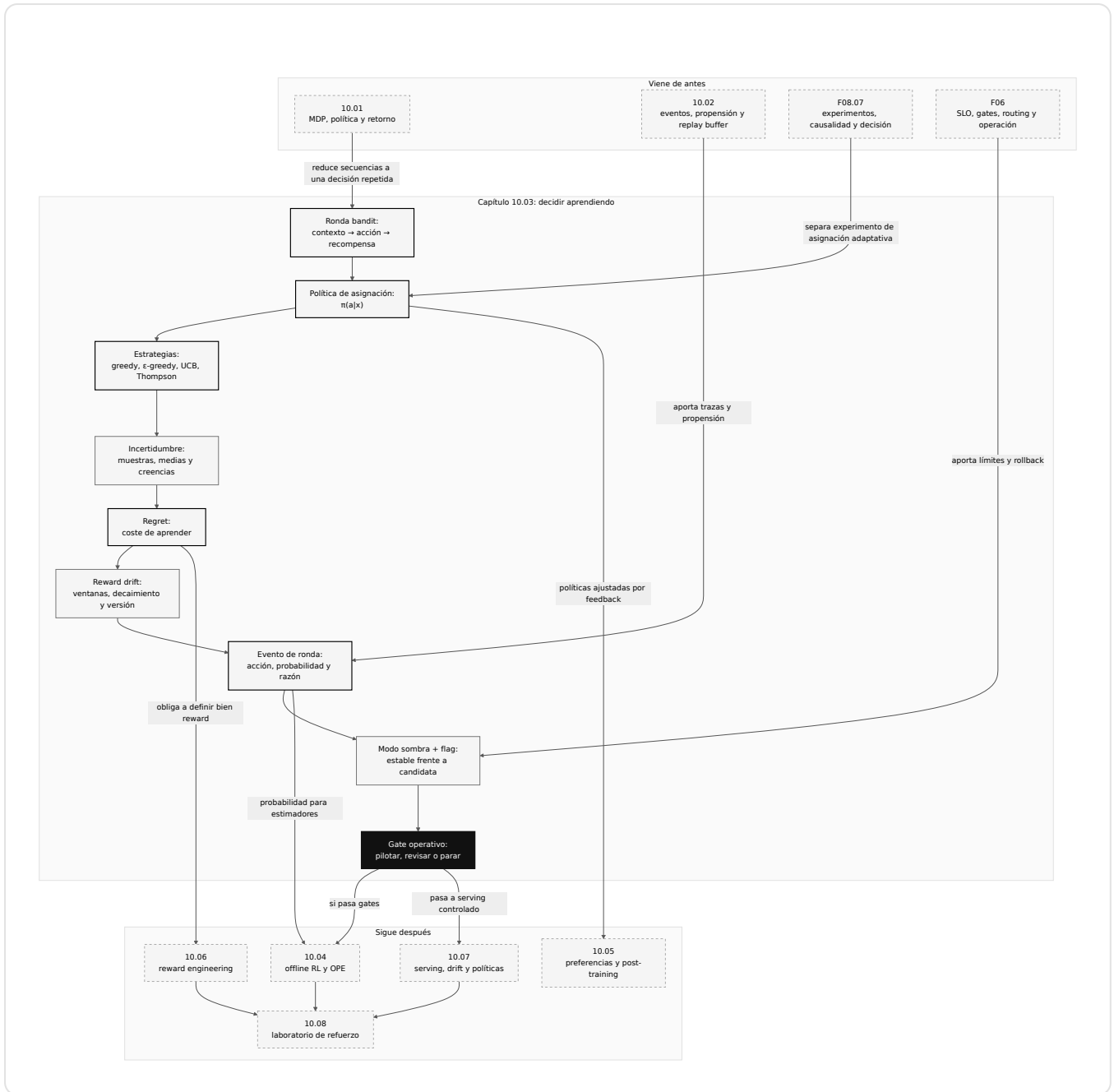
La segunda familia no entrena políticas; gobierna despliegues. OpenFeature aporta una API común y contexto de evaluación. LaunchDarkly, Unleash, Statsig, GrowthBook, Eppo y Optimizely sirven para controlar exposición, segmentación, variantes y lectura de experimentos.¹⁶¹⁷¹⁸¹⁹²⁰ Langfuse y Phoenix ayudan a mirar trazas y evaluaciones de aplicaciones LLM; no sustituyen el gate estadístico, pero hacen visible qué prompt, retrieval o ruta de modelo produjo cada recompensa.²¹²²

Anatomía de una política bandit operable



Cómo encaja todo

Este mapa debe leerse como el paso intermedio entre “tengo datos de interacción” y “puedo evaluar o servir una política”. Del capítulo 01 heredamos política, retorno y valor. Del capítulo 02 heredamos evento, propensión y linaje. Aquí convertimos todo eso en una decisión adaptativa con presupuesto de exploración, regret, modo sombra y gate operativo. Después, el capítulo 04 podrá hacer evaluación offline y el capítulo 07 podrá servir políticas con drift y rollback.



Dónde solía tropezar yo

TROIEZO	POR QUÉ OCURRE	ANTÍDOTO
Ver bandits como A/B testing automático.	Se parecen en superficie.	Separar estimación fija, asignación adaptativa y regret.
Optimizar clic, aceptación o score sin coste.	La recompensa cómoda suele estar disponible antes.	Restar coste, latencia, reapertura y fallos de evidencia.
Explorar en todos los casos.	Parece estadísticamente eficiente.	Definir slices donde la exploración se limita o desactiva.
No guardar propensión.	Solo miramos acción y recompensa.	Registrar $\pi(a x)$, política, semilla y razón de selección.
Confundir mejor media con mejor decisión.	La incertidumbre desaparece en el agregado.	Comparar greedy, ϵ -greedy, UCB y Thompson en simulación.
Publicar sin rollback.	La simulación salió bien.	Exigir feature flag, política de reserva y gate por ventana.

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Bandit	Problema de decisión repetida donde elegimos una acción y observamos recompensa.
Contextual bandit	Bandit que usa contexto observado antes de decidir.
Exploración	Probar acciones inciertas para aprender.
Explotación	Usar la acción que parece mejor con la evidencia actual.
Regret	Coste acumulado de no elegir la mejor acción disponible.
UCB	Estrategia que suma media observada y bonus de incertidumbre.
Thompson sampling	Estrategia que muestrea creencias sobre acciones y elige la muestra más alta.
Presupuesto de exploración	Límite explícito de rondas, tráfico o coste destinado a probar.
No estacionariedad	Situación donde la distribución de recompensa cambia con el tiempo.
Shadow	Fase donde la política recomienda, pero el sistema sigue ejecutando la ruta estable.
Feature flag	Mecanismo para controlar variantes, exposición y rollback sin redesplegar código.

Antes de pasar página

- ¿Qué diferencia hay entre exploración y explotación?
- ¿Por qué greedy puede quedar atrapado?
- ¿Qué mide el regret?
- ¿Qué pregunta responde UCB?
- ¿Cómo se actualiza una Beta en Thompson sampling?
- ¿Qué cambia entre bandit y contextual bandit?
- ¿Por qué una media antigua puede ser peligrosa si cambia el tráfico o el modelo?
- ¿Qué límites pondrías antes de usar bandits para routing de modelos?
- ¿Qué campos guardarías para poder evaluar una política offline después?

- ¿Qué debe ocurrir en shadow antes de permitir un piloto?
- ¿Qué diferencia hay entre una feature flag y una política bandit?

Para saber más

Auer, P., Cesa-Bianchi, N. y Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47, 235-256. <https://doi.org/10.1023/A:1013689704352>

Bubeck, S. y Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1), 1-122. <https://doi.org/10.1561/22000000024>

Chapelle, O. y Li, L. (2011). An empirical evaluation of Thompson sampling. *Advances in Neural Information Processing Systems 24*. <https://papers.nips.cc/paper/4321-an-empirical-evaluation-of-thompson-sampling>

CleanRL. (2026). *CleanRL Documentation*. <https://docs.cleanrl.dev/>

Dudík, M., Langford, J. y Li, L. (2011). Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning*, 1097-1104. https://icml.cc/2011/papers/511_icmlpaper.pdf

Eppo. (2026). *The Eppo Docs*. <https://docs.getepo.com/>

GrowthBook. (2026). *GrowthBook Documentation*. <https://docs.growthbook.io/>

Lai, T. L. y Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1), 4-22. [https://doi.org/10.1016/0196-8858\(85\)90002-8](https://doi.org/10.1016/0196-8858(85)90002-8)

Langfuse. (2026). *Langfuse Documentation*. <https://langfuse.com/docs>

LaunchDarkly. (2026). *Experimentation*. <https://launchdarkly.com/docs/home/experimentation>

LaunchDarkly. (2026). *Releasing features with LaunchDarkly*. <https://launchdarkly.com/docs/home/releases/releasing>

Li, L., Chu, W., Langford, J. y Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th International Conference on World Wide Web*, 661-670. <https://doi.org/10.1145/1772690.1772758>

OpenFeature. (2026). *Evaluation Context*. <https://openfeature.dev/specification/sections/evaluation-context/>

OpenTelemetry. (2026). *Feature flag semantic conventions*. <https://opentelemetry.io/docs/specs/semconv/registry/attributes/feature-flag/>

Optimizely. (2026). *Introduction to Optimizely Feature Experimentation*. <https://docs.developers.optimizely.com/feature-experimentation/docs>

Phoenix. (2026). *Phoenix Documentation*. <https://arize.com/docs/phoenix>

PyTorch. (2026). *TorchRL Documentation*. <https://docs.pytorch.org/rl/>

Ray. (2026). *RLlib: Industry-Grade, Scalable Reinforcement Learning*. <https://docs.ray.io/en/latest/rllib/>

Statsig. (2026). *Experiment Options*. <https://docs.statsig.com/statsig-warehouse-native/features/experiment-options>

Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.^a ed.). MIT Press. <https://incompleteideas.net/book/the-book-2nd.html>

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4), 285-294. <https://doi.org/10.2307/2332286>

Unleash. (2026). *How to perform a gradual rollout*. <https://docs.getunleash.io/feature-flag-tutorials/use-cases/gradual-rollout>

Vowpal Wabbit. (2026). *Vowpal Wabbit Documentation*. <https://vowpalwabbit.org/docs/>

En resumen

IDEA	QUÉ DEBES RECORDAR
Un bandit aprende mientras decide.	Por eso necesita límites operativos, no solo métrica final.
Explorar tiene coste.	El regret mide la factura de aprender.
UCB explora por incertidumbre.	No prueba al azar sin más: prioriza acciones poco observadas.
Thompson sampling muestrea creencias.	Explora proporcionalmente a la incertidumbre posterior.
El contexto cambia la decisión.	Routing de modelos, prompts y RAG suelen ser contextual bandits.
Sin propensión no hay buena evaluación posterior.	Cada ronda debe guardar política, probabilidad, acción y reward.
Las recompensas pueden cambiar.	Usa ventanas, decaimiento y versionado cuando cambian modelos, prompts o tráfico.
Shadow reduce riesgo.	La política recomienda antes de decidir y deja comparar contra la ruta estable.
La feature flag no es el bandit.	La flag controla exposición y rollback; la política decide la acción dentro de los límites.

Notas

1. Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.^a ed.). MIT Press. <https://incompleteideas.net/book/the-book-2nd.html> ↵
2. LaunchDarkly. (2026). *Releasing features with LaunchDarkly*. <https://launchdarkly.com/docs/home/releases/releasing>. Consultado el 8 de junio de 2026. ↵
3. OpenFeature. (2026). *Evaluation Context*. <https://openfeature.dev/specification/sections/evaluation-context/>. Consultado el 8 de junio de 2026. ↵
4. Auer, P., Cesa-Bianchi, N. y Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47, 235-256. <https://doi.org/10.1023/A:1013689704352> ↵
5. Lai, T. L. y Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1), 4-22. [https://doi.org/10.1016/0196-8858\(85\)90002-8](https://doi.org/10.1016/0196-8858(85)90002-8) ↵

6. Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4), 285-294.
<https://doi.org/10.2307/2332286> ↵
7. Chapelle, O. y Li, L. (2011). An empirical evaluation of Thompson sampling. *Advances in Neural Information Processing Systems 24*. <https://papers.nips.cc/paper/4321-an-empirical-evaluation-of-thompson-sampling> ↵
8. Bubeck, S. y Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1), 1-122.
<https://doi.org/10.1561/22000000024> ↵
9. Li, L., Chu, W., Langford, J. y Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th International Conference on World Wide Web*, 661-670. <https://doi.org/10.1145/1772690.1772758> ↵
10. Dudík, M., Langford, J. y Li, L. (2011). Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning*, 1097-1104.
https://icml.cc/2011/papers/511_icmlpaper.pdf ↵
1. OpenTelemetry. (2026). *Feature flag semantic conventions*.
<https://opentelemetry.io/docs/specs/semconv/registry/attributes/feature-flag/>. Consultado el 8 de junio de 2026. ↵
2. Vowpal Wabbit. (2026). *Vowpal Wabbit Documentation*. <https://vowpalwabbit.org/docs/>. Consultado el 7 de junio de 2026. ↵
3. Ray. (2026). *RLlib: Industry-Grade, Scalable Reinforcement Learning*.
<https://docs.ray.io/en/latest/rllib/>. Consultado el 7 de junio de 2026. ↵
4. PyTorch. (2026). *TorchRL Documentation*. <https://docs.pytorch.org/rl/>. Consultado el 7 de junio de 2026. ↵
5. CleanRL. (2026). *CleanRL Documentation*. <https://docs.cleanrl.dev/>. Consultado el 7 de junio de 2026. ↵
6. Unleash. (2026). *How to perform a gradual rollout*. <https://docs.getunleash.io/feature-flag-tutorials/use-cases/gradual-rollout>. Consultado el 8 de junio de 2026. ↵
7. Statsig. (2026). *Experiment Options*. <https://docs.statsig.com/statsig-warehouse-native/features/experiment-options>. Consultado el 8 de junio de 2026. ↵
8. GrowthBook. (2026). *GrowthBook Documentation*. <https://docs.growthbook.io/>. Consultado el 8 de junio de 2026. ↵
9. Eppo. (2026). *The Eppo Docs*. <https://docs.geteppo.com/>. Consultado el 8 de junio de 2026. ↵

- !0. Optimizely. (2026). *Introduction to Optimizely Feature Experimentation*. <https://docs.developers.optimizely.com/feature-experimentation/docs>. Consultado el 8 de junio de 2026. ↵
 - !1. Langfuse. (2026). *Langfuse Documentation*. <https://langfuse.com/docs>. Consultado el 8 de junio de 2026. ↵
 - !2. Arize AI. (2026). *Phoenix Documentation*. <https://arize.com/docs/phoenix>. Consultado el 8 de junio de 2026. ↵
-

Cuadernos para practicar

Escanea el código con el móvil para abrir el cuaderno en Google Colab y ejecutarlo sin instalar nada.



Explorar o explotar: el dilema de los bandidos

<https://colab.research.google.com/github/686f6c61/iaparagentecuriosa-notebooks/blob/main/Facs%C3%ADmil%2010/03-bandits-explorar-explotar.ipynb>

Capítulo 04: Offline RL y evaluación contrafactual de políticas

La pregunta incómoda antes de mover tráfico

Imagina que ya tienes lo que construimos en los capítulos anteriores: un dataset de eventos con estado, acción, recompensa, política histórica, propensión y trazas. También tienes una política candidata que, según simulación, parece repartir mejor el tráfico entre modelo rápido, modelo fuerte y revisión humana. La tentación es lanzarla a un 5 % de usuarios y mirar.

Offline RL y OPE aparecen justo para frenar esa prisa. La pregunta ya no es “¿funcionó la política que estaba en producción?”. La pregunta es más difícil:

¿qué habría pasado si otra política hubiera tomado las decisiones, usando solo los datos que ya tenemos?

Esa pregunta es contrafactual. Para una misma consulta observamos una acción y una recompensa. No observamos la recompensa de las acciones que no se eligieron. En el facsímil 8 vimos que este es el corazón del razonamiento causal: tenemos un resultado observado y varios resultados posibles que no llegaron a ocurrir.

Offline RL intenta aprender una política desde un dataset fijo. OPE, *off-policy evaluation*, intenta estimar el valor de una política nueva con datos generados por otra política. Las dos piezas están relacionadas, pero no son lo mismo. Puedes hacer OPE sin entrenar una política nueva. Y puedes entrenar offline RL sin tener aún suficiente confianza para desplegar.

Qué no es offline RL

Offline RL no es “entrenar con logs” sin más. Un log de producto puede estar lleno de acciones históricas, pero si no contiene propensión, acciones disponibles, recompensa versionada, contexto suficiente y cobertura, no es un dataset defendible de RL.

Tampoco es aprendizaje supervisado con un nombre más sofisticado. En supervisado solemos aprender de etiquetas observadas: entrada, etiqueta, pérdida. En offline RL, la etiqueta no es “la acción correcta”. La acción del log fue la acción que tomó una política histórica, con sus

sesgos, sus límites y sus zonas ciegas. Copiarla puede ser una baseline útil, pero no necesariamente una mejora.

Y OPE no es una garantía absoluta. Un estimador puede decir que una política candidata parece mejor, pero esa estimación vive bajo supuestos: soporte suficiente, propensiones correctas, reward fiable, independencia razonable, ausencia de cambios fuertes de distribución y varianza controlada. Si alguno de esos supuestos falla, la cifra puede ser tranquilizadora y estar equivocada.

Tres preguntas distintas

Antes de elegir método, conviene separar tres preguntas:

PREGUNTA	NOMBRE	QUÉ RESPONDE	QUÉ NO RESPONDE
¿Cuánto valdría una política candidata con datos históricos?	OPE	Estima valor sin ejecutarla.	No crea una política nueva.
¿Qué política puedo aprender de un dataset fijo?	Offline RL	Entrena o selecciona una política sin interacción nueva.	No garantiza que el dataset cubra buenas acciones.
¿Puedo pasar a modo sombra o piloto?	Gate operativo	Decide si la evidencia basta para el siguiente paso.	No sustituye monitorización online.

En un proyecto de IA, OPE suele ser el primer freno serio antes de publicar una política adaptativa. Offline RL viene después si quieres aprender una política más compleja a partir de experiencias históricas. El gate operativo decide si la evidencia es suficiente para avanzar o si toca volver al dataset.

El dataset de OPE como producto de datos

Un ingeniero de datos no debería tratar OPE como un notebook suelto. Debe tratarlo como un producto de datos versionado. La unidad mínima no es “una fila con reward”, sino un evento de decisión con linaje suficiente para reconstruir quién decidió, qué podía decidir, con qué probabilidad y qué recompensa se le atribuyó después.

CAMPO	POR QUÉ ES OBLIGATORIO	QUÉ FALLO EVITA
<code>event_id</code>	Identifica la decisión.	Duplicados invisibles.
<code>occurred_at</code>	Sitúa la decisión en el tiempo.	Mezclar políticas o rewards de épocas distintas.
<code>context</code>	Describe el estado observado antes de actuar.	Evaluar una política con información que no tenía al decidir.
<code>allowed_actions</code>	Catálogo de acciones disponibles en ese momento.	Comparar contra acciones imposibles.
<code>action</code>	Acción realmente ejecutada.	No hay evento evaluable sin acción observada.
<code>behavior_policy_id</code>	Política histórica que produjo el dato.	Tratar logs como si fueran neutrales.
<code>behavior_action_probability</code>	Propensión histórica.	No poder corregir sesgo de asignación.
<code>target_policy_probability_by_action</code>	Probabilidades de la política candidata sobre el mismo contexto.	No poder estimar contrafactualmente.
<code>reward</code>	Resultado atribuido a la decisión.	Optimizar métricas cómodas pero irrelevantes.
<code>reward_version</code>	Versión de la función de recompensa.	Mezclar definiciones incompatibles.
<code>q_model_version</code>	Versión del modelo usado por DM/DR/FQE.	No reproducir estimaciones.
<code>dataset_snapshot_id</code>	Corte congelado del dataset.	Recalcular sobre datos cambiantes sin saberlo.

La tabla anterior no es burocracia. Es ingeniería de reproducibilidad. Si cambias `reward_version`, un mismo evento puede valer otra cosa. Si cambias `behavior_policy_id`, las propensiones ya no significan lo mismo. Si pierdes `allowed_actions`, quizá evalúas una política candidata que “elige” una acción que en producción no habría estado disponible.

En un warehouse, conviene separar al menos tres tablas:

TABLA	CONTENIDO	QUIÉN LA CONSUME
rl_ope_events	Eventos históricos con contexto, acción, propensión, reward y versiones.	Data engineering, ML, auditoría.
rl_ope_importance_weights	Pesos por evento y run de evaluación.	ML, analítica, revisión técnica.
rl_ope_runs	Resultado agregado: estimadores, intervalos, ESS y status.	Comité técnico, CI/CD, operación.

El esquema mínimo de OPE no pretende imponer Postgres, BigQuery o Snowflake; pretende fijar el contrato mental: OPE deja evidencias consultables, no solo una gráfica en un notebook.

Particionar sin hacerse trampas

El error clásico en OPE es usar todo para todo. Entrenas un modelo de recompensa con los mismos eventos que luego evalúas, ajustas umbrales mirando el resultado final y acabas creyendo que el estimador confirmó una política cuando en realidad has filtrado información del futuro.

Una partición defendible suele separar:

CORTE	USO	REGLA PRÁCTICA
Entrenamiento del modelo $\hat{q}$	Aprender reward esperado o función Q.	Datos anteriores en el tiempo.
Validación de $\hat{q}$	Medir calibración y error del modelo auxiliar.	Corte temporal posterior, sin tocar política candidata.
OPE final	Estimar política candidata y aplicar gate.	Snapshot congelado, sin reentrenar después de mirar el resultado.
Shadow o piloto	Confirmar online con exposición controlada.	Solo si OPE no bloquea.

La regla de oro es temporal: si en producción no sabías algo en el momento de decidir, no puede aparecer en el contexto de entrenamiento ni en el modelo de evaluación. En datos de producto esto incluye rewards tardíos, reaberturas, tickets escalados, costes finales, feedback humano y cambios de política.

El valor de una política

En el capítulo 01 definimos retorno. Para evaluar una política π , queremos su valor esperado:

$$V(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \gamma^t r_t \right]$$

SÍMBOL O	SIGNIFICADO	EJEMPLO
$V(\pi)$	Valor esperado de ejecutar la política π .	Recompensa media esperada del routing candidato.
τ	Trayectoria completa de estados, acciones y recompensas.	Secuencia de decisiones sobre un ticket.
γ	Factor de descuento.	0,95
r_t	Recompensa en el paso t .	Calidad menos coste y penalización por reapertura.
T	Horizonte del episodio.	5 decisiones del asistente.

El problema offline es que los datos no vienen de π . Vienen de una política histórica b , llamada política de comportamiento. Nuestro dataset se parece a:

$$\mathcal{D} = \{\tau_i\}_{i=1}^n, \quad \tau_i \sim b$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\mathcal{D}$	Dataset histórico fijo.	Logs de 12.000 tickets.
n	Número de trayectorias o eventos.	12.000
b	Política de comportamiento que generó los datos.	Routing estable anterior.
π	Política objetivo que queremos evaluar.	Routing candidato.

La diferencia entre b y π lo cambia todo. Si la política candidata elige acciones que la histórica casi nunca eligió, estamos intentando adivinar fuera de los datos. Ahí empieza el peligro.

Propensión: el campo que decide si podemos evaluar

La propensión es la probabilidad con la que la política histórica eligió la acción observada:

$$b(a_t \mid s_t)$$

La política candidata tiene su propia probabilidad:

$$\pi(a_t \mid s_t)$$

Con ambas podemos construir el peso de importancia por paso:

$$\rho_t = \frac{\pi(a_t \mid s_t)}{b(a_t \mid s_t)}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$b(a_t \mid s_t)$	Probabilidad histórica de la acción observada.	0,25
$\pi(a_t \mid s_t)$	Probabilidad candidata para esa misma acción.	0,50
ρ_t	Peso de importancia.	2,0

Si $\rho_t = 2$, ese evento cuenta el doble porque la política candidata habría elegido esa acción con más frecuencia que la histórica. Si $\rho_t = 0,1$, cuenta poco. Si $b(a_t \mid s_t)$ es casi cero, el peso explota. Y si $b(a_t \mid s_t) = 0$ pero $\pi(a_t \mid s_t) > 0$, no hay estimación estadística honesta: la política candidata quiere hacer algo que el dataset no observó.

La condición de soporte (o cobertura) es uno de los supuestos de base de la evaluación fuera de política: si la política candidata quiere elegir una acción, el dato histórico debe contener evidencia para esa acción en ese estado. Se escribe así:

$$\pi(a \mid s) > 0 \Rightarrow b(a \mid s) > 0$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\pi(a s) > 0$	La política candidata puede elegir esa acción.	Quiere usar <code>revision_humana</code> .
$b(a s) > 0$	La política histórica también la eligió alguna vez con probabilidad positiva.	Hay tickets parecidos con revisión humana registrada.
$\Rightarrow$	Implicación.	Si quieres evaluarla, debe existir soporte.

Esta es una de las frases más importantes del capítulo: no puedes evaluar bien una política que se sale de la cobertura de tus datos.

IPS: corregir por probabilidad histórica

El estimador más directo es *inverse propensity scoring* o importance sampling. En su versión por evento:

$$\widehat{V_{\text{IPS}}}(\pi) = \frac{1}{n} \sum_{i=1}^n \frac{\pi(a_i | x_i)}{b(a_i | x_i)} r_i$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\widehat{V_{\text{IPS}}}$	Estimación IPS del valor de la política candidata.	0,789
$\widehat{x_i}$	Contexto observado.	Criticidad, idioma, complejidad.
a_i	Acción registrada.	<code>modelo_fuerte</code> .
r_i	Recompensa observada.	0,79
n	Número de eventos.	12

Ejemplo numérico: si la política histórica eligió `modelo_fuerte` con probabilidad 0,28, la candidata lo habría elegido con probabilidad 0,42, y la recompensa observada fue 0,76, el término IPS es:

$$\frac{0,42}{0,28} \cdot 0,76 = 1,14$$

Ese evento empuja hacia arriba la estimación. No porque su recompensa tenga una propiedad especial, sino porque la candidata habría elegido esa acción con más frecuencia. El coste es la varianza: unos pocos pesos grandes pueden dominar la media.

WIS: normalizar los pesos

Weighted importance sampling normaliza por la suma de pesos:

$$V_{\text{WIS}}(\pi) = \frac{\sum_{i=1}^n w_i r_i}{\sum_{i=1}^n w_i}, \quad w_i = \frac{\pi(a_i \mid x_i)}{b(a_i \mid x_i)}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
w_i	Peso de importancia del evento i .	1,5
$\sum w_i r_i$	Recompensas ponderadas.	8,75
$\sum w_i$	Peso total acumulado.	11,68
V_{WIS}	Estimación normalizada.	0,749

WIS suele ser más estable que IPS cuando los pesos no suman cerca de n . Pero también introduce sesgo. No hay almuerzo gratis: IPS puede ser insesgado y ruidoso; WIS puede ser más estable y sesgado. Por eso no miramos un único número.

Tamaño efectivo de muestra

Contar eventos no basta. Si tienes 10.000 filas, pero la política candidata depende de 20 eventos con pesos enormes, tu muestra efectiva es pequeña.

El tamaño efectivo de muestra se calcula a menudo como:

$$\text{ESS} = \frac{(\sum_{i=1}^n w_i)^2}{\sum_{i=1}^n w_i^2}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
ESS	Tamaño efectivo de muestra.	11,02
w_i	Peso de importancia.	1,54
n	Número de eventos reales.	12

En el cuaderno de este capítulo, el escenario sano tiene 12 eventos y ESS cercano a 11. El escenario roto tiene solo 6 eventos y pesos máximos por encima de 30. Ahí una cifra de reward alta no es una buena noticia: es una alarma.

Intervalos de confianza: no publiques una media desnuda

Una estimación puntual puede ser engañosa. Si `doubly_robust=0,744`, la pregunta de ingeniería es: ¿esa cifra es estable o depende demasiado de unas pocas filas? Una forma simple de introducir incertidumbre es bootstrap: re-muestrear el dataset con reemplazo muchas veces y recalcular el estimador.

Para B re-muestreos, obtenemos:

$$V_{\text{DR}}^{(1)}, V_{\text{DR}}^{(2)}, \dots, V_{\text{DR}}^{(B)}$$

Después tomamos percentiles. Para un intervalo del 90 %:

$$[P_5(V_{\text{DR}}), P_{95}(V_{\text{DR}})]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
B	Número de re-muestreos bootstrap.	500
$V_{\text{DR}}^{(b)}$	Estimación DR en el re-muestreo b .	0,731
$\widehat{P}_5$	Percentil 5 de las estimaciones.	0,700
P_{95}	Percentil 95 de las estimaciones.	0,786

Esto no convierte OPE en verdad absoluta. Pero mejora la conversación: si el límite inferior del intervalo queda por debajo del umbral de calidad, no deberíamos avanzar aunque la media sea bonita. En decisiones profesionales, el gate debería mirar al menos media, intervalo, ESS y soporte por slice.

Direct method y doubly robust

Otra familia usa un modelo de recompensa $\hat{q}(x, a)$ que predice qué reward esperamos para cada acción:

$$\widehat{V_{DM}(\pi)} = \frac{1}{n} \sum_{i=1}^n \sum_a \pi(a \mid x_i) \hat{q}(x_i, a)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\hat{q}(x_i, a)$	Reward predicho para acción a en contexto x_i .	0,78
$\sum_a \pi(a \mid x_i) \hat{q}(x_i, a)$	Valor esperado según el modelo.	0,73
V_{DM}	Estimación por modelo directo.	0,733

El riesgo es obvio: si $\hat{q}$ está mal, DM hereda ese error con mucha confianza.

Doubly robust combina el modelo con la corrección por propensión:

$$\widehat{V_{DR}(\pi)} = \frac{1}{n} \sum_{i=1}^n \left[\sum_a \pi(a \mid x_i) \hat{q}(x_i, a) + \frac{\pi(a_i \mid x_i)}{b(a_i \mid x_i)} (r_i - \hat{q}(x_i, a_i)) \right]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
V_{DR}	Estimación doubly robust.	0,744
$\hat{q}(x_i, a_i)$	Predicción del reward para la acción observada.	0,74
$r_i - \hat{q}(x_i, a_i)$	Error residual del modelo.	0,02
$\frac{\pi}{b}$	Corrección por diferencia entre políticas.	1,5

Jiang y Li extendieron estimadores doubly robust a evaluación off-policy secuencial en RL, buscando reducir varianza frente a importance sampling puro.¹ Dudík, Langford y Li ya habían trabajado estimadores doubly robust para evaluación y aprendizaje de políticas en bandits.² La lectura práctica: DR no te libra de malos datos, pero suele dar una señal más resistente cuando el modelo de reward y la propensión aportan información complementaria.

FQE: evaluar aprendiendo una función Q

Fitted Q Evaluation, FQE, intenta estimar el valor de una política fija aprendiendo una función $Q^\pi(s, a)$. No está aprendiendo una política nueva; está evaluando una política objetivo. La actualización conceptual se parece a Bellman:

$$\hat{Q}_{k+1}(s_t, a_t) \leftarrow r_t + \gamma \mathbb{E}_{a' \sim \pi(\cdot | s_{t+1})} [\hat{Q}_k(s_{t+1}, a')]$$

SÍMBOL O	SIGNIFICADO	EJEMPLO
$\hat{Q}_k$	Aproximación de Q en la iteración k .	Modelo auxiliar de valor.
s_t, a_t, r_t, s_{t+1}	Transición observada.	Estado, acción, reward y siguiente estado de una trayectoria.
$\pi(\cdot s_{t+1})$	Política que estamos evaluando.	Routing candidato.
γ	Descuento temporal.	0,95

FQE es útil cuando tienes trayectorias y un espacio donde un modelo Q puede generalizar. Pero esa generalización es justo su riesgo: si el modelo aprende valores para zonas fuera del soporte, puede sonar convincente y estar extrapolando. Por eso herramientas como d3rlpy lo presentan dentro de OPE, no como sustituto de auditoría de datos.³

Una lectura práctica:

SI TIENES...	FQE AYUDA A...	CUIDADO
Trayectorias completas	Evaluar valor secuencial, no solo eventos sueltos.	Necesitas s_{t+1} , terminales y rewards bien atribuidos.
Acciones discretas	Comparar políticas candidatas con una Q aproximada.	El espacio debe estar cubierto.
Modelo Q validado	Reducir dependencia de pesos extremos.	Si Q está mal calibrado, el estimador se vuelve confiado.

Cuando la decisión es secuencial

Hasta ahora hemos usado una versión por evento porque es más fácil de llevar a código. En RL secuencial, una trayectoria completa tiene pesos acumulados:

$$W_i = \prod_{t=0}^T \frac{\pi(a_{i,t} \mid s_{i,t})}{b(a_{i,t} \mid s_{i,t})}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
W_i	Peso total de la trayectoria i .	3,8
$a_{i,t}$	Acción de la trayectoria i en paso t .	Llamar herramienta.
$s_{i,t}$	Estado de la trayectoria i en paso t .	Ticket con adjunto y prioridad media.
Π	Producto de ratios por paso.	Multiplicar $1,2 \cdot 0,8 \cdot 1,5$

El producto es peligroso. Si cada paso tiene algo de varianza, multiplicar ratios durante muchos pasos puede explotar. Por eso OPE secuencial es difícil y por eso los datos del capítulo 02 importan tanto: sin trayectorias limpias, tiempos correctos y reward atribuible, el estimador se convierte en una ilusión.

Thomas y Brunskill propusieron métodos de evaluación off-policy orientados a eficiencia de datos y confianza, dentro de esta preocupación por estimar políticas con datos históricos de forma útil para decisiones reales.⁴

Offline RL: aprender sin pedir más interacción

OPE evalúa una política candidata. Offline RL intenta aprender una política desde un dataset fijo. Levine, Kumar, Tucker y Fu describen offline RL como una familia de algoritmos que usan datos previamente recogidos sin nueva recolección online.⁵

El problema técnico central es el desplazamiento de distribución. Un algoritmo puede asignar valor alto a acciones que casi no aparecen en el dataset. Esa sobreestimación puede producir políticas que parecen excelentes en entrenamiento y fallan cuando se ejecutan.

Hay varias familias de solución:

FAMILIA	IDEA	CUÁNDO ENCAJA	RIESGO
Behavior cloning	Imitar la política histórica.	Baseline fuerte si los datos son buenos.	No mejora más allá de lo observado.
Restricción de soporte	Mantener la política cerca de acciones vistas.	Cuando salirse del dataset es peligroso.	Puede ser demasiado conservadora.
Q conservador	Penalizar valores altos fuera del dato.	Cuando la sobreestimación es el problema principal.	Ajustar conservadurismo no es trivial.
Fitted Q Evaluation	Aprender $\hat{Q}$ para evaluar una política.	OPE con función de valor.	Depende de generalización del modelo.
Model-based offline RL	Aprender dinámica o reward y simular.	Entornos con transiciones modelables.	Error de modelo acumulado.

BCQ restringe acciones candidatas para evitar extrapolar fuera del batch.⁶ CQL propone aprender funciones Q conservadoras que reduzcan el valor de acciones fuera de distribución.⁷ D4RL ayudó a estandarizar benchmarks para este escenario con datasets de políticas, demostraciones y mezclas.⁸

La idea de CQL (Kumar y colaboradores, 2020) se puede leer así:

$$\min_Q \text{BellmanError}(Q) + \alpha (\mathbb{E}_{a \sim \pi}[Q(s, a)] - \mathbb{E}_{a \sim \mathcal{D}}[Q(s, a)])$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$Q(s, a)$	Valor estimado de acción en estado.	Valor de usar <code>modelo_fuerte</code> .
BellmanError	Error frente a la consistencia de Bellman.	Diferencia entre target y predicción.
α	Peso del término conservador.	0,5
$a \sim \pi$	Acciones que propone la política aprendida.	Acciones candidatas nuevas.
$a \sim \mathcal{D}$	Acciones presentes en el dataset.	Acciones históricas observadas.

No hace falta memorizar el objetivo. Quédate con la intuición: si una acción aparece poco o nada en los datos, no dejes que el modelo le asigne valor alto sin evidencia.

Ejemplos que sí se parecen al trabajo real

Routing de modelos

Tienes logs de una política estable que elegía entre modelo rápido, modelo fuerte y revisión humana. Quieres saber si una política candidata, más inclinada al modelo fuerte en casos complejos, habría mejorado calidad sin disparar coste.

La evaluación no debería mirar solo reward medio. Debe mirar:

SEÑAL	POR QUÉ IMPORTA
Propensión histórica	Sin ella no corriges el sesgo de asignación.
Soporte por slice	Si la candidata elige <code>revision_humana</code> en slices donde apenas hay ejemplos, no hay base suficiente.
ESS	Si unos pocos eventos dominan, la conclusión es frágil.
Gap IPS-WIS	Si se separan, los pesos están pesando demasiado.
Gap DM-DR	Si se separan, el modelo de reward o las propensiones no están alineadas.

RAG adaptativo

La acción puede ser `top_k=3` , `top_k=6` , `hybrid_search` o `rerank` . La política histórica quizá usaba casi siempre `top_k=6` . La candidata quiere usar reranking cuando detecta consulta ambigua. Si en los logs casi no hay reranking en consultas ambiguas, OPE no puede prometer que esa decisión funcione. Puede sugerir “necesitas más datos de esa zona”.

Agentes con herramientas

Una política decide si el agente responde, busca en documentos, llama a una API o pide revisión. Offline RL puede ser tentador porque hay muchas trazas. Pero si la política histórica no registró acciones disponibles, permisos, resultado de herramientas y reward tardío, el dataset no permite saber qué habría pasado con otra secuencia.

Diagnóstico por slice y acción

La media global suele mentir por omisión. Una política puede tener buen DR agregado y estar mal soportada en `alta_criticidad` . O puede tener soporte para `modelo_fuerte` , pero no para `revision_humana` . En IA aplicada, ese detalle importa más que la cifra final.

El cuaderno del capítulo genera dos artefactos nuevos:

ARTEFACTO	PREGUNTA QUE RESPONDE
<code>output/slice_diagnostics.cs</code> v	¿Qué pasa por slice: DR, ESS, peso máximo y soporte?
<code>output/support_matrix.csv</code>	¿Qué acciones tienen masa de política candidata pero poco soporte observado?

Un patrón sano se parece a esto:

SEÑAL	LECTURA SANA	LECTURA PELIGROSA
ESS por slice	Cada slice conserva muestra efectiva razonable.	Un slice queda dominado por uno o dos eventos.
Peso máximo	Ningún evento manda la estimación.	Un evento tiene peso enorme y reward alto.
Soporte de acción	Las acciones con probabilidad candidata aparecen en logs.	La candidata quiere acciones no observadas.
DR por slice	No hay caída fuerte oculta.	La media global sube, pero un slice crítico cae.

Para ingeniería de datos, esto se convierte en un control de calidad programable: si `support_matrix.csv` muestra `has_observed_support=false` para una acción con masa candidata relevante, no se arregla con una frase optimista. Se arregla recogiendo datos, limitando la política candidata o cambiando el alcance del piloto.

El matiz importante es “masa relevante”. Si una política candidata asigna un 0,5 % a una acción no observada, quizá basta con documentarlo y mantener el piloto limitado. Si asigna un 30 %, el gate debe bloquear. Por eso el contrato incluye `max_unsupported_target_probability_mass` : no basta saber si hay huecos, hay que medir cuánto peso de decisión cae en ellos.

Qué mira un equipo antes de aprobar OPE

Antes de permitir modo sombra, un equipo serio debería poder responder:

REVISIÓN	PREGUNTA CONCRETA	EVIDENCIA
Contrato de datos	¿Están todos los campos obligatorios y sus versiones?	<code>ope_contract.json</code> y schema de warehouse.
Linaje	¿Qué snapshot, política, reward y modelo Q produjeron la estimación?	<code>dataset_snapshot_id</code> , <code>reward_version</code> , <code>q_model_version</code> .
Soporte	¿La candidata decide dentro de zonas observadas?	<code>support_matrix.csv</code> .
Incertidumbre	¿El límite inferior bootstrap pasa el umbral?	<code>bootstrap_ci_lower</code> .
Estabilidad	¿IPS, WIS, DM y DR cuentan una historia parecida?	<code>estimator_scorecard.csv</code> .
Riesgo por slice	¿Hay slices que bloquean aunque el promedio pase?	<code>slice_diagnostics.csv</code> .
Operación	¿Qué ocurre si el modo sombra contradice OPE?	Runbook del capítulo 10.03 y capítulo 10.07.

La salida profesional no es “mi notebook dice que mejora”. La salida profesional es una tarjeta de calidad: qué se evaluó, con qué snapshot, qué supuestos pasan, qué supuestos fallan y qué decisión permite.

Herramientas y estado del arte

Fecha de corte: 8 de junio de 2026. Fuentes consultadas: literatura clásica de OPE/offline RL, d3rlpy, SCOPE-RL, Minari, RLlib y TorchRL. Lo estable es el problema: soporte, propensión, distribución y estimación contrafactual. Lo cambiante son APIs, benchmarks y librerías.

HERRAMIENTA	QUÉ APORTA	QUÉ MIRAR ANTES
d3rlpy	Librería offline RL con APIs tipo scikit-learn, algoritmos offline y OPE/FQE.	Si tus datos caben en su formato y si los algoritmos soportan tu espacio de acciones.
SCOPE-RL	Flujo para offline RL, OPE y selección de políticas.	Métricas de riesgo-retorno, estimadores disponibles y diseño experimental.
Minari	API/datasets para offline RL dentro del ecosistema Farama.	Formato de datasets, versionado y si necesitas reproducir benchmarks tipo D4RL.
Ray RLlib	Entrenamiento RL a escala y APIs offline sobre Ray Data.	Complejidad operativa, integración con datos y si necesitas escala real.
TorchRL	Objetivos offline como CQL dentro de PyTorch.	Si quieres control de tensores, pérdidas y pipelines de investigación.

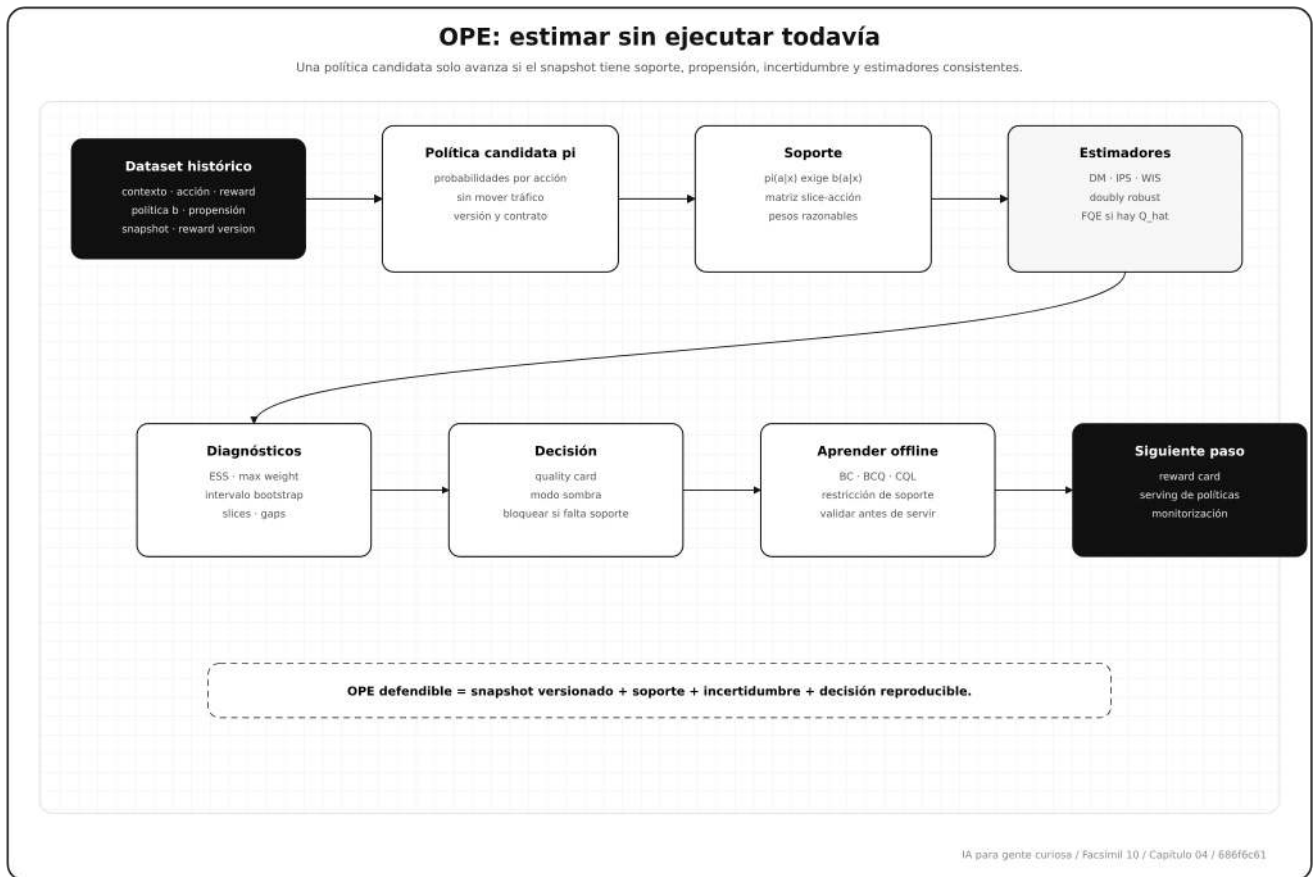
d3rlpy documenta OPE con Fitted Q Evaluation; SCOPE-RL cubre offline RL, OPE y selección de políticas; Minari organiza datasets de offline RL; RLlib documenta APIs offline; TorchRL incluye objetivos offline como CQL.[910111213](#)

La elección práctica no empieza por “qué librería está de moda”. Empieza por el formato del dato. Minari es interesante cuando quieres datasets versionados con una API de offline RL. RLlib encaja si ya usas Ray Data y necesitas ingestión a escala; su documentación offline se apoya en flujos de lectura/escritura de datos como Parquet. d3rlpy es más directo para experimentar con algoritmos y OPE/FQE. SCOPE-RL ayuda a ordenar el flujo completo de offline RL, OPE y selección de política, incluyendo escenarios de high-confidence OPE en sus ejemplos. TorchRL es útil si el equipo quiere mantener control bajo en PyTorch.

Para un equipo de datos, la pregunta de integración sería:

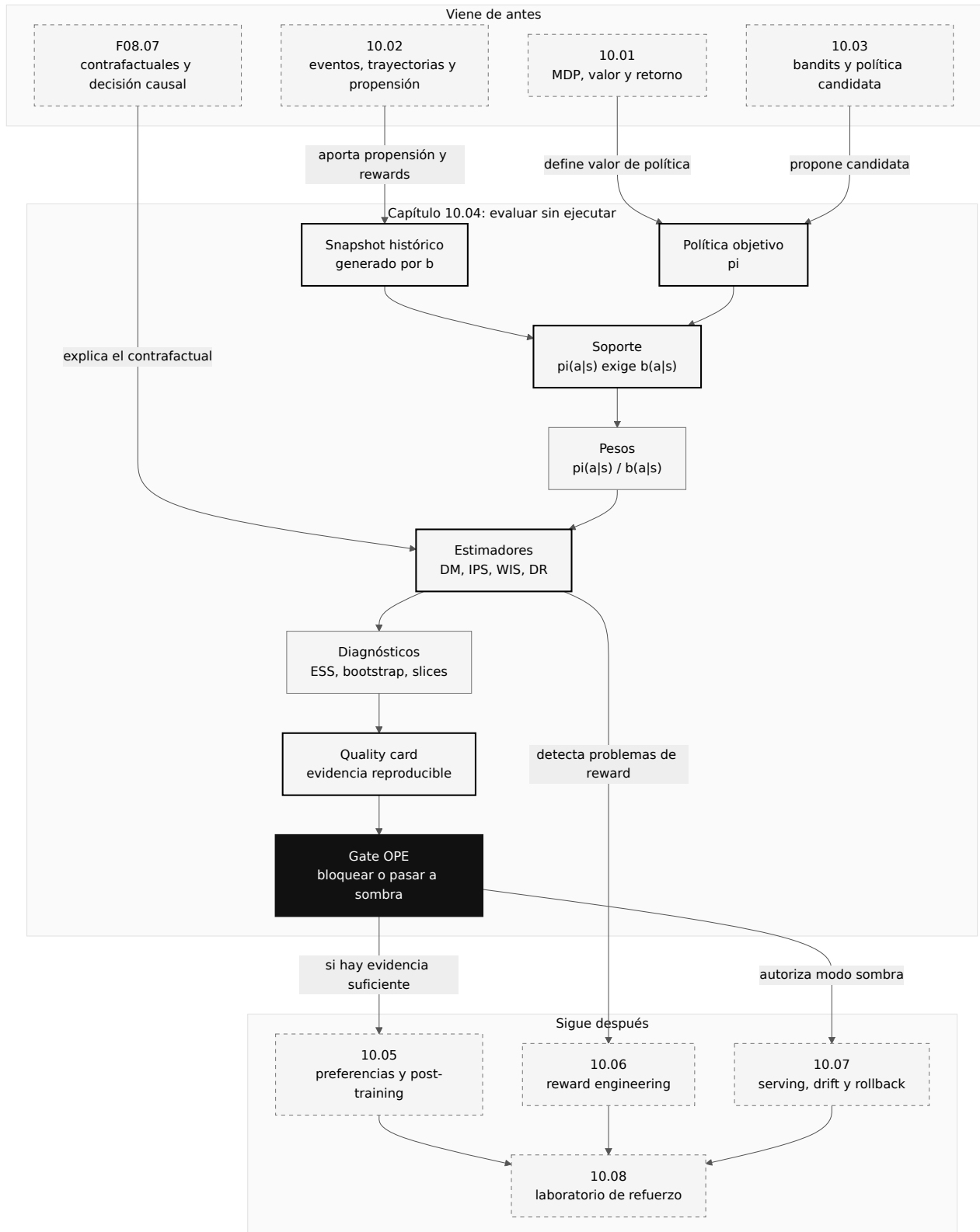
PREGUNTA	POR QUÉ IMPORTA
¿Puedo cargar mi snapshot sin perder <code>behavior_action_probability</code> ?	Sin propensión, OPE se debilita.
¿Puedo versionar dataset, reward y política candidata?	Sin versiones, no hay reproducibilidad.
¿Puedo exportar pesos y diagnósticos al warehouse?	Sin evidencias consultables, no hay revisión operativa.
¿Puedo separar train/validation/OPE por tiempo?	Sin partición temporal, hay riesgo de leakage.
¿Puedo bloquear por slice?	Sin slices, la media global oculta fallos.

Anatomía de una evaluación offline defendible



Cómo encaja todo

Este mapa une el facsímil como una cadena de evidencia. El capítulo 02 nos da logs con propensión. El capítulo 03 nos da políticas candidatas y validación online limitada. Aquí preguntamos si una política puede evaluarse con datos históricos antes de ejecutarla. Lo que salga de este capítulo alimenta post-training, diseño de reward y serving de políticas.



Dónde solía tropezar yo

TROIEZO	POR QUÉ OCURRE	ANTÍDOTO
Llamar offline RL a cualquier entrenamiento con logs.	El nombre suena amplio.	Exigir estado, acción, reward, propensión, acciones disponibles y versión.
Creer que OPE demuestra que una política funcionará.	Devuelve una cifra.	Leer soporte, ESS, pesos máximos y gaps entre estimadores.
Olvidar la política de comportamiento.	Los logs parecen datos neutros.	Guardar <code>behavior_policy_id</code> y <code>behavior_action_probability</code> .
Aceptar pesos enormes.	IPS puede dar un valor alto.	Revisar distribución de pesos y bloquear si unos pocos eventos mandan.
Entrenar una política fuera del soporte.	El modelo generaliza con confianza.	Usar restricciones, conservadurismo o recoger datos de cobertura.
Mezclar rewards de versiones distintas.	Es cómodo juntar históricos.	Versionar reward y evaluar por cortes temporales.

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Offline RL	Aprender políticas desde un dataset fijo de experiencias.
OPE	Evaluar una política candidata con datos generados por otra política.
Política de comportamiento	Política histórica que produjo los datos.
Política objetivo	Política candidata que queremos evaluar.
Propensión	Probabilidad histórica de la acción observada.
Importance sampling	Reponderación por la relación entre política candidata e histórica.
WIS	Importance sampling normalizado por suma de pesos.
Direct method	Estimador que usa un modelo de reward para predecir valor.
Doubly robust	Estimador que combina modelo de reward y corrección por propensión.
ESS	Tamaño efectivo de muestra tras aplicar pesos.
Intervalo bootstrap	Rango de estimaciones obtenido al re-muestrear el dataset.
FQE	Estimación offline de una función Q para evaluar una política fija.
Soporte	Cobertura suficiente para evaluar acciones que la política candidata podría elegir.
Quality card	Documento que resume estimadores, checks, slices, soporte y decisión.
CQL	Método offline RL que penaliza valores altos fuera del dato observado.

Antes de pasar página

- ¿Qué diferencia hay entre OPE y offline RL?
- ¿Qué es la política de comportamiento?
- ¿Por qué `action_probability` es tan importante?
- ¿Qué significa que una política candidata esté fuera de soporte?
- ¿Qué mide IPS?

- ¿Por qué WIS puede ser más estable que IPS?
- ¿Qué combina doubly robust?
- ¿Qué te dice ESS que no te dice el número bruto de eventos?
- ¿Por qué mirarías el límite inferior del intervalo bootstrap?
- ¿Qué diferencia hay entre una media global y un diagnóstico por slice?
- ¿Qué función cumple una quality card de OPE?
- ¿Por qué OPE puede autorizar modo sombra, pero no producción amplia?
- ¿Qué artefactos produce el cuaderno del capítulo?

Para saber más

d3rlpy. (2026). *Off-Policy Evaluation*.

https://d3rlpy.readthedocs.io/en/v2.4.0/references/off_policy_evaluation.html

Dudík, M., Langford, J. y Li, L. (2011). Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning*, 1097-1104.

https://icml.cc/2011/papers/511_icmlpaper.pdf

Farama Foundation. (2026). *Minari Documentation*. <https://minari.farama.org/>

Fu, J., Kumar, A., Nachum, O., Tucker, G. y Levine, S. (2020). D4RL: Datasets for deep data-driven reinforcement learning. *arXiv:2004.07219*. <https://arxiv.org/abs/2004.07219>

Fujimoto, S., Meger, D. y Precup, D. (2019). Off-policy deep reinforcement learning without exploration. *Proceedings of the 36th International Conference on Machine Learning*, 97, 2052-2062. <https://proceedings.mlr.press/v97/fujimoto19a.html>

Jiang, N. y Li, L. (2016). Doubly robust off-policy value evaluation for reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning*, 48, 652-661. <https://proceedings.mlr.press/v48/jiang16.html>

Kumar, A., Zhou, A., Tucker, G. y Levine, S. (2020). Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 1179-1191. <https://arxiv.org/abs/2006.04779>

Levine, S., Kumar, A., Tucker, G. y Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv:2005.01643*. <https://arxiv.org/abs/2005.01643>

PyTorch. (2026). *Offline RL Methods*.

https://docs.pytorch.org/rl/main/reference/objectives_offline.html

Ray. (2026). *Offline RL API*. https://docs.ray.io/en/latest/rllib/package_ref/offline.html

SCOPE-RL. (2026). *SCOPE-RL Documentation*. <https://scope-rl.readthedocs.io/en/latest/documentation/index.html>

Seno, T. y Imai, M. (2022). d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315), 1-20. <https://jmlr.org/papers/v23/22-0017.html>

Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.^a ed.). MIT Press. <https://incompleteideas.net/book/the-book-2nd.html>

Thomas, P. S. y Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning*, 48, 2139-2148. <https://arxiv.org/abs/1604.00923>

En resumen

IDEA	QUÉ DEBES RECORDAR
OPE estima una política no ejecutada.	Usa datos de otra política y necesita propensión.
Offline RL aprende desde un dataset fijo.	El peligro central es salirse del soporte del dato.
IPS corrige por probabilidad.	Puede tener varianza enorme si hay pesos grandes.
WIS estabiliza.	Normaliza pesos, pero puede introducir sesgo.
Doubly robust combina dos fuentes.	Modelo de reward más corrección por propensión.
ESS importa más que contar filas.	Muchas filas pueden equivaler a pocas si los pesos dominan.
Los intervalos importan.	Una media sin incertidumbre no basta para decidir.
Los slices importan.	Una política puede pasar en promedio y fallar donde más duele.
OPE es un producto de datos.	Necesita snapshot, warehouse, contrato, linaje y quality card.
El gate decide el siguiente paso.	Un buen OPE permite modo sombra o piloto limitado, no confianza ciega.

Notas

1. Jiang, N. y Li, L. (2016). Doubly robust off-policy value evaluation for reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning*, 48, 652-661. <https://proceedings.mlr.press/v48/jiang16.html> ↵
2. Dudík, M., Langford, J. y Li, L. (2011). Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning*, 1097-1104. https://icml.cc/2011/papers/511_icmlpaper.pdf ↵
3. d3rlpy. (2026). *Off-Policy Evaluation*. https://d3rlpy.readthedocs.io/en/v2.4.0/references/off_policy_evaluation.html. Consultado el 8 de junio de 2026. ↵
4. Thomas, P. S. y Brunskill, E. (2016). Data-efficient off-policy policy evaluation for reinforcement learning. *Proceedings of the 33rd International Conference on Machine Learning*, 48, 2139-2148. <https://arxiv.org/abs/1604.00923> ↵
5. Levine, S., Kumar, A., Tucker, G. y Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv:2005.01643*. <https://arxiv.org/abs/2005.01643> ↵
6. Fujimoto, S., Meger, D. y Precup, D. (2019). Off-policy deep reinforcement learning without exploration. *Proceedings of the 36th International Conference on Machine Learning*, 97, 2052-2062. <https://proceedings.mlr.press/v97/fujimoto19a.html> ↵
7. Kumar, A., Zhou, A., Tucker, G. y Levine, S. (2020). Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 1179-1191. <https://arxiv.org/abs/2006.04779> ↵
8. Fu, J., Kumar, A., Nachum, O., Tucker, G. y Levine, S. (2020). D4RL: Datasets for deep data-driven reinforcement learning. *arXiv:2004.07219*. <https://arxiv.org/abs/2004.07219> ↵
9. d3rlpy. (2026). *Off-Policy Evaluation*. https://d3rlpy.readthedocs.io/en/v2.4.0/references/off_policy_evaluation.html. Consultado el 8 de junio de 2026. ↵
10. SCOPE-RL. (2026). *SCOPE-RL Documentation*. <https://scope-rl.readthedocs.io/en/latest/documentation/index.html>. Consultado el 8 de junio de 2026. ↵
11. Farama Foundation. (2026). *Minari Documentation*. <https://minari.farama.org/>. Consultado el 8 de junio de 2026. ↵
12. Ray. (2026). *Offline RL API*. https://docs.ray.io/en/latest/rllib/package_ref/offline.html. Consultado el 8 de junio de 2026. ↵

.3. PyTorch. (2026). *Offline RL Methods*.

https://docs.pytorch.org/rl/main/reference/objectives_offline.html. Consultado el 8 de junio de 2026. ↵

Capítulo 05: Preferencias y post-training: RLHF, DPO, RLAIIF y RLVR

El modelo ya sabe escribir; ahora queremos que decida mejor

Un LLM pre-entrenado aprende una distribución de texto. Eso no significa que se comporte como necesitamos en un producto: puede contestar con formato incorrecto, citar sin evidencia suficiente, usar una herramienta cuando no toca, alargar una respuesta sencilla o preferir una solución elegante pero poco verificable. El post-training aparece cuando queremos convertir capacidad general en comportamiento operativo.

La diferencia importante es esta: no estamos "metiendo conocimiento" sin más. Estamos cambiando qué salidas se vuelven más probables bajo una señal. Esa señal puede venir de demostraciones humanas, pares de preferencias, evaluadores automáticos, tests, verificadores simbólicos o reglas de negocio. Si la señal es pobre, el modelo no aprende nuestra intención; aprende la señal pobre con mucha seguridad.

Ouyang y otros popularizaron una receta moderna para modelos instructivos: demostraciones para SFT, rankings humanos, reward model y optimización con RLHF. En su distribución de evaluación, un InstructGPT de 1,3B parámetros fue preferido frente a GPT-3 175B, lo que ilustra que el comportamiento post-entrenado puede pesar más que el tamaño bruto para ciertas tareas.¹ Antes, Stiennon y otros ya habían mostrado el esquema de comparaciones humanas, reward model y fine-tuning de una política para resumen, con análisis de generalización del reward model.²

En este capítulo no vamos a vender nombres. Vamos a leer cada método como un contrato de ingeniería:

SI TIENES...	PUEDES PLANTEAR...	LO QUE DE VERDAD ESTÁS OPTIMIZANDO
Entradas y salidas correctas	SFT	Imitación de demostraciones.
Para el mismo prompt, una respuesta preferida y otra rechazada	DPO, IPO, ORPO	Margen relativo entre salidas.
Comparaciones humanas suficientes	RLHF clásico	Recompensa aprendida y política ajustada.
Principios escritos y evaluación asistida por otro modelo	RLAIF / Constitutional AI	Crítica, revisión y preferencia escalada.
Tests, resultados exactos o graders reproducibles	RFT / RLVR / GRPO	Recompensa programática o verificable.

La palabra clave es "señal". Un equipo serio no empieza preguntando "¿hacemos RLHF?". Empieza preguntando "¿qué señal tenemos, cómo la medimos, qué sesgos tiene, qué coste añade y qué evidencia deja?".

Qué no es post-training

Post-training no es pedirle al modelo en el prompt que "sea mejor". Un prompt puede orientar una respuesta concreta, pero no cambia los pesos. Si cada llamada tiene que recordar diez páginas de normas para producir un JSON estable, quizá necesitas contrato de salida, RAG, evaluaciones o fine-tuning; no una frase más larga.

Tampoco es una forma barata de actualizar conocimiento vivo. Si cambia cada día el catálogo, el precio, la política de un cliente o la documentación interna, RAG o herramientas suelen encajar mejor. Fine-tuning y preferencias enseñan comportamiento repetido: formato, criterio, estilo, uso de herramientas, rechazo de respuestas insuficientes, razonamiento verificable. No son una base de datos.

Y no es una garantía ética automática. Si entrenas preferencias de mala calidad, el sistema puede volverse más convincente sin volverse más correcto. Si entrenas con un verificador incompleto, puede aprender el camino que pasa el verificador, no necesariamente el comportamiento profesional que querías.

La taxonomía que usaría un ingeniero

Hay varias familias, y conviene separarlas por el tipo de dato que exigen:

FAMILIA	DATO MÍNIMO	ENTRENAMIENTO TÍPICO	QUÉ ARTEFACTO DEBERÍAS GUARDAR
SFT	prompt , completion o conversación completa.	Cross-entropy sobre la respuesta objetivo.	Dataset de demostraciones, plantilla de chat, eval de formato.
Reward modeling	prompt , chosen , rejected .	Modelo escalar que predice preferencia.	Reward card, acuerdo entre anotadores, métricas por slice.
RLHF con PPO	Prompts, reward model, política referencia, rollout.	Muestrear respuestas y optimizar reward con anclaje KL.	Config de PPO, curvas de reward/KL, samples y evals.
DPO/IPO/ORPO	Pares preferido/rechazado.	Pérdida contrastiva offline.	Auditoría de pares, márgenes chosen/rejected, eval retenida.
KTO	Etiquetas binarias deseable/no deseable, no siempre pareadas.	Pérdida inspirada en utilidad humana.	Distribución de etiquetas, balance por tarea, revisión de negativos.
RLAIF	Críticas o preferencias generadas por IA bajo principios.	Similar a preferencias, pero con evaluador automático.	Principios, prompt del evaluador, checks humanos de muestra.
RLVR/RF T/GRPO	Prompts y reward verificable por programa o grader.	Generación online y actualización por recompensa.	Código del grader, casos ocultos, tasa de error del grader.

DPO no es "SFT con dos columnas más". RLHF no es "DPO caro". RLVR no es "DPO con tests". La diferencia práctica está en cuándo se generan las respuestas, quién asigna la señal, si hay modelo de recompensa separado, si hay política de referencia y si el entrenamiento necesita rollouts durante el proceso.

El dato de preferencias como producto de datos

Un par de preferencias parece sencillo:

```
{
  "prompt": "Resume esta incidencia y propone siguiente paso.",
  "chosen": "Resumen breve, causa probable y acción concreta.",
  "rejected": "Texto largo, sin acción y con datos no comprobados."
}
```

Pero un dataset profesional necesita bastante más:

CAMPO	POR QUÉ IMPORTA	FALLO QUE EVITA
<code>pair_id</code>	Identificador estable.	Duplicados invisibles.
<code>prompt_id</code>	Agrupar pares del mismo caso.	Mezclar ejemplos sin saber su origen.
<code>task_family</code>	Clasifica el tipo de tarea.	Que una media oculte fallos por dominio.
<code>prompt</code>	Entrada real enviada al modelo.	Entrenar con contexto que no existirá en producción.
<code>chosen</code>	Respuesta preferida.	Señal positiva.
<code>rejected</code>	Respuesta menos preferida.	Señal contrastiva.
<code>preference_reason</code>	Razón de la elección.	Pares correctos por azar pero sin criterio.
<code>rubric_scores</code>	Calidad, evidencia, formato, coste, etc.	Saber qué dimensión explica la preferencia.
<code>annotator_ids</code>	Identifica evaluadores o fuentes.	No detectar desacuerdos sistemáticos.
<code>agreement</code>	Acuerdo entre evaluadores.	Entrenar sobre pares ambiguos como si fueran claros.
<code>source_policy</code>	Modelo o sistema que generó cada respuesta.	Filtrar ventajas de una política concreta.
<code>created_at</code>	Fecha del ejemplo.	Mezclar normas antiguas y nuevas.
<code>reference_answer</code>	Respuesta o criterio esperado si existe.	No poder revisar el par.
<code>verifier_result</code>	Resultado de test o grader si aplica.	Confundir preferencia subjetiva con verificación objetiva.

La columna más ignorada suele ser `preference_reason`. Sin razón, el par dice "A gana a B", pero no dice si gana por ser correcto, breve, educado, trazable, barato, seguro, completo o simplemente más bonito. En entrenamiento, todas esas razones se mezclan en una sola dirección de gradiente.

Bradley-Terry: convertir comparaciones en probabilidad

Una forma clásica de modelar preferencias pareadas es Bradley-Terry. Si tenemos un prompt x , una respuesta ganadora y_w y una respuesta perdedora y_l , el modelo de recompensa asigna puntuaciones $r_\phi(x, y_w)$ y $r_\phi(x, y_l)$. La probabilidad de que y_w sea preferida puede escribirse así:

$$P(y_w \succ y_l \mid x) = \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x	Prompt o contexto.	Ticket de soporte con SLA y logs.
y_w	Respuesta preferida.	Diagnóstico con acción y evidencia.
y_l	Respuesta rechazada.	Respuesta genérica sin evidencia.
$r_\phi(x, y)$	Reward model con parámetros ϕ .	Puntuación escalar de una respuesta.
σ	Sigmoide.	Convierte diferencia en probabilidad.
$\succ$	"Preferido a".	y_w gana a y_l .

La pérdida del reward model suele empujar a que la respuesta preferida tenga más puntuación:

$$\mathcal{L}_{RM} = -\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))$$

Si $r_\phi(x, y_w) = 3,1$ y $r_\phi(x, y_l) = 1,7$, la diferencia es 1,4. La sigmoide de 1,4 es aproximadamente 0,80. El reward model está diciendo: "según lo que he aprendido, hay un 80 % de probabilidad de que la primera respuesta sea la preferida". No está diciendo que sea verdadera, ni que sea legal, ni que sea adecuada para todos los usuarios. Solo traduce la señal entrenada.

Ese matiz evita mucha confusión. Un reward model no es una autoridad. Es un estimador entrenado sobre preferencias observadas.

RLHF como pipeline de ingeniería

La receta clásica de RLHF se puede escribir así:

modelo base

- > SFT con demostraciones
- > generación de respuestas candidatas
- > comparaciones humanas
- > reward model
- > optimización de política
- > evaluación retenida y gates

El paso de RL suele optimizar una política $\pi_{\theta}(y \mid x)$ para maximizar recompensa, pero evitando alejarse demasiado de una política de referencia π_{ref} . El objetivo de esta fase, en la formulación habitual de RLHF, es la recompensa esperada con una penalización KL que frena el alejamiento de la referencia (la optimización suele hacerse con PPO, citado más abajo):

$$\max_{\theta} \mathbb{E}_{y \sim \pi_{\theta}(\cdot \mid x)} [r_{\phi}(x, y) - \beta \text{KL}(\pi_{\theta}(\cdot \mid x) \parallel \pi_{\text{ref}}(\cdot \mid x))]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
π_{θ}	Política/modelo que ajustamos.	Modelo SFT que queremos mejorar.
π_{ref}	Política de referencia.	Copia congelada del modelo antes de RL.
$r_{\phi}(x, y)$	Reward model.	Puntuación de preferencia estimada.
KL	Divergencia entre distribuciones.	Cuánto se aleja el modelo ajustado.
β	Peso de la penalización KL.	Freno contra cambios bruscos.

PPO, presentado por Schulman y otros, es una familia de métodos de gradiente de política que alterna muestreo y varias épocas de actualización con un objetivo sustituto.³ En RLHF, la parte complicada no es solo "usar PPO". Es mantener coherentes cuatro piezas: política, referencia, reward model y datos de rollout.

SEÑAL EN UNA RUN DE RLHF	QUÉ DEBERÍAS MIRAR	QUÉ INDICA SI SE TUERCE
<code>train_reward_mean</code>	Recompensa media en entrenamiento.	Puede subir por aprender un atajo de la señal.
<code>valid_reward_mean</code>	Recompensa en validación.	Si no sube, el entrenamiento memoriza patrones locales.
<code>kl_to_ref</code>	Distancia a la referencia.	Si crece demasiado, el modelo cambia más de lo previsto.
<code>response_length</code>	Longitud de salida.	Reward puede estar premiando verbosidad.
<code>format_pass_rate</code>	Cumplimiento de contrato de salida.	Calidad aparente con formato roto.
<code>human_eval_delta</code>	Preferencia humana retenida.	El reward model no basta como prueba final.

OpenAI documenta el RFT actual como un proceso basado en graders programables que puntúan respuestas candidatas, con métricas de reward por entrenamiento y validación; a fecha de corte, su documentación también indica que la plataforma de fine-tuning se está cerrando para nuevos usuarios, por lo que aquí lo usamos como referencia conceptual y operativa, no como recomendación universal de disponibilidad.⁴

DPO: preferencia sin montar un ciclo RL completo

DPO parte del mismo tipo de dato básico:

```
prompt x
respuesta elegida y_w
respuesta rechazada y_l
modelo referencia pi_ref
modelo que entrenamos pi_theta
```

La intuición es subir la probabilidad relativa de la respuesta preferida frente a la rechazada, comparando el cambio respecto al modelo de referencia. Rafailov y otros formularon DPO como una forma de optimizar preferencias con una pérdida de clasificación, sin entrenar explícitamente un reward model separado ni hacer muestreo de RL durante el ajuste.⁵

Una forma habitual de escribir la pérdida:

$$\mathcal{L}_{\text{DPO}} = -\log \sigma \left(\beta [\log \pi_{\theta}(y_w | x) - \log \pi_{\theta}(y_l | x) - \log \pi_{\text{ref}}(y_w | x) + \log \pi_{\text{ref}}(y_l | x)] \right)$$

SÍMBOL O	SIGNIFICADO	LECTURA PRÁCTICA
$\pi_{\theta}(y_w x)$	Probabilidad de la respuesta elegida bajo el modelo entrenado.	Queremos que suba frente a la rechazada.
$\pi_{\theta}(y_l x)$	Probabilidad de la respuesta rechazada.	Queremos que baje relativamente.
π_{ref}	Modelo de referencia.	Ancla el cambio.
β	Intensidad del contraste.	Más alto suele endurecer la preferencia.
σ	Sigmoide.	Convierte margen en probabilidad de acierto.

No hace falta memorizar la fórmula completa. La lectura de ingeniería es:

DPO aprende el margen entre una salida preferida y una no preferida, condicionado por un modelo de referencia.

La ventaja operativa es clara: dataset offline, sin reward model separado y sin infraestructura RL completa. El riesgo también: si los pares tienen desacuerdo, ruido, sesgo de longitud o razones mezcladas, DPO lo aprende. Herramientas como TRL esperan datasets de preferencia con `prompt` , `chosen` y `rejected` , y documentan métricas como log-probabilidades, recompensas implícitas, márgenes y accuracy de recompensa.⁶

IPO, KTO y ORPO: no todo es DPO

En la práctica moderna aparecen variantes porque DPO no agota el problema.

MÉT ODO	QUÉ CAMBIA	CUÁNDO LO MIRARÍA	CUIDADO
IPO	Reinterpreta preferencias dentro de una familia teórica más general.	Cuando preocupa saturación o sobreconfianza del margen.	No elimina la necesidad de buenos pares.
KTO	Aprende de señales binarias deseable/no deseable, no necesariamente pareadas.	Cuando tienes feedback de producto tipo pulgar arriba/abajo.	Balance de positivos/negativos y definición de "deseable".
ORPO	Combina SFT y preferencia con odds ratio sin modelo referencia separado.	Cuando quieres un flujo simple con pares chosen/rejected.	Sensible al dataset y a la mezcla entre imitación y contraste.

KTO se apoya en teoría de prospecto y propone aprender desde una señal binaria de deseabilidad, lo que puede encajar con feedback de producto que no viene en pares perfectos.⁷ ORPO propone una optimización por odds ratio que integra preferencia durante el SFT, sin una fase adicional con modelo de referencia.⁸

La decisión no debería basarse en el nombre del método, sino en el dato disponible:

DATO REAL EN TU EMPRESA	MÉTODO CANDIDATO	ANTES DE ENTRENAR
Conversaciones corregidas por expertos	SFT	Revisar plantilla de chat y retener casos difíciles.
Pares de respuestas para el mismo prompt	DPO u ORPO	Auditar duplicados, desacuerdo y longitud.
Likes/dislikes sueltos	KTO	Separar señal de satisfacción de señal de verdad.
Tests o graders robustos	RFT/RLVR/GRPO	Versionar grader y casos ocultos.
Principios y revisión asistida	RLAIF	Auditar el evaluador automático con muestra humana.

RLAIF: feedback de IA, pero con contrato

Constitutional AI propuso usar principios escritos para generar críticas, revisiones y feedback desde IA, reduciendo dependencia directa de preferencias humanas en determinadas fases.⁹ La idea no es "que otro modelo tenga razón". La idea es hacer explícito el criterio, escalar

revisión y dejar trazabilidad.

Un flujo RLAIIF defendible necesita:

PIEZA	PREGUNTA DE INGENIERÍA
Principios	¿Qué regla concreta decide que una respuesta es mejor?
Prompt del evaluador	¿Se puede reproducir la crítica?
Modelo evaluador	¿Qué versión se usó y con qué parámetros?
Muestra revisada por personas	¿Cuánto coincide el evaluador automático con criterio humano?
Dataset generado	¿Qué pares se aceptan, cuáles se descartan y por qué?
Evals retenidas	¿Mejora fuera del dataset generado por el evaluador?

RLAIIF puede ser útil cuando el volumen de revisión humana no escala o cuando quieres aplicar principios repetibles. Pero si no guardas el prompt del evaluador, la versión del modelo y la muestra de control, has creado una caja opaca más.

RLVR y RFT: cuando la recompensa se puede comprobar

RLVR, reinforcement learning from verifiable rewards, es especialmente interesante en código, matemáticas, SQL, razonamiento con respuesta exacta, extracción estructurada y tareas donde un verificador puede asignar una recompensa reproducible. DeepSeek-R1 mostró una línea influyente: entrenar razonamiento con RL y recompensas verificables en tareas como matemáticas, código y STEM, con razonamiento emergente sin depender siempre de trayectorias humanas etiquetadas.¹⁰

La recompensa verificable no es siempre binaria. Puede componerse:

$$R(y) = \lambda_1 R_{\text{exactitud}}(y) + \lambda_2 R_{\text{formato}}(y) + \lambda_3 R_{\text{evidencia}}(y) - \lambda_4 C_{\text{coste}}(y)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$R(y)$	Recompensa total de la respuesta.	0,84
$R_{\text{exactitud}}$	Si la respuesta resuelve la tarea.	Tests pasan.
R_{formato}	Si cumple el contrato de salida.	JSON válido con campos obligatorios.
$R_{\text{evidencia}}$	Si cita o usa pruebas correctas.	IDs de documentos recuperados.
C_{coste}	Penalización por tokens, latencia o pasos.	Respuesta demasiado larga.
λ_i	Peso de cada componente.	0,5, 0,2, 0,2, 0,1

Un grader puede ser un `string_check`, una similitud textual, un evaluador por modelo, una ejecución Python o una combinación ponderada. OpenAI documenta graders con salida de 0 a 1 y multigraders para componer varias señales; lo relevante para nosotros es la idea de contrato de recompensa versionado.¹¹

Ejemplos donde RLVR tiene sentido:

DOMINIO	GRADER ÚTIL	LO QUE FALTA SI SOLO MIRAS EL GRADER
Código	Tests unitarios, lint, tipos, benchmark.	Casos ocultos, rendimiento y mantenibilidad.
SQL	Ejecutar contra fixtures y comparar filas.	Datos reales, coste de consulta y permisos.
RAG	Verificar que la respuesta cita fragmentos recuperados.	Síntesis correcta y ausencia de inferencias no soportadas.
Matemáticas	Resultado exacto o verificador simbólico.	Robustez ante enunciados ambiguos.
Operación de agentes	Simulador de herramientas y contrato de permisos.	Cambios de estado reales y rollback.

El peligro no desaparece. Cambia de sitio: ahora vive en el grader. Si el grader es incompleto, la política aprende a optimizar esa parte visible. Por eso el capítulo 06 entra después: reward engineering no es decorar una métrica, es diseñar una señal que aguante presión.

El bucle completo de post-training

Un bucle profesional suele tener estas fases:

1. Definir comportamiento objetivo con ejemplos y contraejemplos.
2. Elegir señal: demostración, preferencia, reward model, grader o combinación.
3. Diseñar dataset con linaje, slices y contrato.
4. Separar train, validación, test retenido y revisión humana.
5. Entrenar con método compatible con el dato.
6. Medir métricas internas del método: loss, reward, KL, margen, accuracy, longitud, coste.
7. Medir métricas externas: tareas reales, regresiones, formato, evidencia, latencia, satisfacción.
8. Revisar samples, no solo números.
9. Congelar versión y publicar tarjeta técnica.
10. Servir con monitorización, rollback y drift.

Esto conecta directamente con el capítulo 04. Allí preguntábamos si una política candidata podía evaluarse offline. Aquí preguntamos si una política lingüística puede moverse con preferencias o recompensas sin perder trazabilidad. En ambos casos, el núcleo es el mismo: no basta entrenar; hay que poder defender la evidencia.

Un caso completo: asistente RAG interno

Supón que tienes un asistente RAG para políticas internas. La versión actual responde con citas, pero a veces mezcla documentos, contesta sin evidencia suficiente o devuelve JSON roto cuando el sistema aguas abajo espera campos obligatorios. El equipo quiere "mejorarlo con post-training". Antes de elegir método, escribimos el caso como ingeniería:

PIEZA	DECISIÓN CONCRETA
Tarea	Contestar dudas de política interna con cita y abstención si no hay evidencia.
Baseline	Prompt actual + RAG + validador JSON.
Modelo base	Modelo instructivo ya servido en producción o candidato abierto con licencia compatible.
Dato disponible	Conversaciones corregidas, pares <code>chosen/rejected</code> , trazas de citas y validaciones JSON.
Señal principal	Preferencia pareada: una respuesta gana por citar bien, respetar formato y no inventar.
Señal auxiliar	Grader de formato JSON y verificador de cita contra fragmentos recuperados.
Método razonable	Primero SFT si el formato falla mucho; después DPO si hay pares limpios.
Método que evitaría de entrada	RLHF/PPO si no hay reward model validado ni infraestructura para rollouts.
Gate mínimo	Mejora sobre baseline en cita válida, exactitud revisada, formato, coste y regresiones.

Un par sano no dice solo "esta respuesta me gusta más". Dice por qué gana:

```
{
  "prompt_id": "rag_policy_042",
  "task_family": "rag",
  "prompt": "¿Puedo pedir reembolso de taxi si salgo tarde de una reunión?",
  "chosen": "Según POL-TRAVEL-04, el reembolso de taxi nocturno se permite si el viaje termina después de las 22:00 y se adjunta recibo. No encuentro excepción para viajes personales.",
  "rejected": "Sí, normalmente la empresa permite taxis si sales tarde y tienes recibo.",
  "preference_reason": "La elegida cita documento, condiciona la respuesta y no extrapola.",
  "rubric_scores": {
    "correctness": 0.89,
    "evidence": 0.94,
    "format": 0.80,
    "cost": 0.76
  }
}
```

La decisión técnica podría ser:

OBSERVACIÓN	CONSECUENCIA
El formato JSON falla un 18 % en baseline.	Primero corregir con prompt, validador o SFT corto.
Hay 3.000 pares limpios con razón y acuerdo alto.	DPO es razonable como siguiente experimento.
Solo 300 pares tienen verificador de cita.	No usar RLVR como señal principal todavía.
El coste de inferencia del modelo ajustado sube 35 %.	La mejora debe aparecer en casos de alto valor, no solo en media.
Hay caída en consultas de RR. HH.	Bloquear publicación aunque la media global mejore.

Este ejemplo enseña una idea importante: post-training no sustituye al RAG, al validador, a la observabilidad ni al diseño de producto. Los conecta.

Guía de anotación: cómo escribir pares que enseñan algo

Una persona anotadora no debería limitarse a elegir A o B. Necesita una rúbrica. Si no, el dataset mezcla gustos, longitud, estilo, corrección y paciencia del día.

REGLA	CÓMO SE APLICA	QUÉ EVITA
Una preferencia debe tener razón	<code>chosen</code> gana por exactitud, evidencia, formato, coste o abstención correcta.	Pares que solo reflejan gusto personal.
El prompt debe ser realista	Mismo contexto que verá el sistema en producción.	Entrenar con información que el modelo no tendrá.
La respuesta rechazada debe ser plausible	No usar respuestas absurdamente malas si la tarea real es difícil.	Inflar métricas con pares demasiado fáciles.
Separar dimensiones	Puntuaciones de verdad, evidencia, formato y coste.	No saber qué aprendió el modelo.
Registrar desacuerdo	Dos personas pueden discrepar; ese dato importa.	Convertir ambigüedad en etiqueta dura.
Revisar longitud	Una respuesta larga no es mejor por defecto.	Sesgo a verbosidad.
Mantener trazabilidad	Fuente, modelo generador, fecha, versión de rúbrica y slice.	No poder reproducir el dataset.

Un par debería descartarse si:

MOTIVO	EJEMPLO
No hay diferencia clara	Ambas respuestas son correctas y equivalentes.
La razón no se puede escribir	"Suenan mejor" sin criterio operativo.
La preferida contradice un verificador	JSON inválido, SQL que no ejecuta, cita ausente.
Falta contexto de producción	El prompt no incluye documentos, permisos o contrato de salida.
Hay desacuerdo fuerte sin resolución	Acuerdo bajo y sin tercera revisión.

Para hacerlo práctico, conviene mantener una guía de anotación junto al dataset. Esa guía no es decoración: el dataset no se entiende sin sus reglas de anotación.

Decidir método también es decidir coste

La tabla siguiente es la que pondría delante de un equipo antes de abrir una máquina con GPU:

MÉTODO	DATO NECESARIO	COSTE RELATIVO	COMPLEJIDAD	RIESGO PRINCIPAL	CUÁNDO LO USARÍA
Prompt + contrato	Instrucciones, ejemplos y validación.	Bajo	Baja	No cambia comportamiento de fondo.	Primer intento, salida estructurada, reglas simples.
SFT	Demostraciones buenas.	Medio	Media	Imitar errores del dataset.	Formato, estilo, tarea repetida y estable.
DPO	Pares chosen/rejected limpios.	Medio	Media	Aprender pares ruidosos con seguridad.	Preferencias claras sin infraestructura RL.
ORPO	Pares y deseo de flujo compacto.	Medio	Media	Mezclar imitación y contraste sin diagnóstico.	Cuando se quiere una fase única supervisada + preferencia.
KTO	Señal binaria deseable/no deseable.	Medio	Media	Feedback binario mal calibrado.	Likes/dislikes, revisiones de producto no pareadas.
Reward model	Rankings o pares suficientes.	Alto	Alta	Reward que puntúa bien en validación pobre.	Antes de RLHF o ranking interno.
RLHF/PP O	Reward model, rollouts, referencia y eval fuerte.	Alto	Alta	Optimizar reward con drift de comportamiento.	Cuando hay equipo, infraestructura y reward validado.
RLVR/G RPO	Grader reproducible y casos retenidos.	Alto	Alta	Aprender el checker en vez de la tarea.	Código, SQL, matemáticas, formato verificable.

Un criterio útil:

$$\text{valor neto} = \Delta Q_{\text{retenida}} - \lambda_c \Delta C_{\text{inferencia}} - \lambda_r R_{\text{regresión}} - \lambda_o C_{\text{operación}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\Delta Q_{\text{retenida}}$	Mejora de calidad en eval no usada para entrenar.	+0,07 en cita correcta.
$\Delta C_{\text{inferencia}}$	Incremento de coste por respuesta.	+35 %.
$R_{\text{regresión}}$	Penalización por caída en slices críticos.	RR. HH. cae 0,04.
$C_{\text{operación}}$	Coste de servir, monitorizar y mantener.	GPU, logs, rollback.
λ	Peso de cada penalización.	Define tolerancia del negocio.

No es una fórmula universal. Es una forma de obligarnos a hablar de calidad, coste y regresiones en la misma frase.

Configuración mínima de referencia

Este bloque no pretende ser una receta para copiar sin pensar. Pretende enseñar qué piezas aparecen en una configuración realista de DPO con LoRA. Si alguien no entiende estos campos, todavía no debería entrenar.

```
run_name: f10-c05-rag-dpo-lora
method: dpo

model:
  base_model: proveedor/modelo-instructivo
  tokenizer: proveedor/modelo-instructivo
  chat_template: versionada_en_repo
  torch_dtype: bfloat16

dataset:
  train_path: data/preference_pairs_train.jsonl
  eval_path: data/preference_pairs_eval.jsonl
  fields:
    prompt: prompt
    chosen: chosen
    rejected: rejected
  max_prompt_length: 2048
  max_completion_length: 768
  split_rule: temporal_por_created_at

lora:
  enabled: true
  r: 16
  alpha: 32
  dropout: 0.05
  target_modules: [q_proj, k_proj, v_proj, o_proj]

training:
  learning_rate: 0.000005
  beta: 0.1
  batch_size: 4
  gradient_accumulation_steps: 8
  epochs: 1
  warmup_ratio: 0.03
  seed: 41

eval:
  every_steps: 100
  metrics:
    - chosen_reward_margin
    - chosen_accuracy
    - format_pass_rate
    - citation_supported_rate
    - avg_completion_tokens
    - latency_p95

gates:
  min_chosen_accuracy: 0.72
  min_citation_supported_rate: 0.78
  max_format_failure_rate: 0.04
```

```
max_avg_token_increase: 0.20
block_on_slice_regression: true
```

La parte que más suele faltar es `gates`. Mucha gente configura entrenamiento, pero no configura una decisión. Entrenar sin gate es producir un checkpoint; no producir una mejora.

Cuando esto sale mal en proyectos reales

SÍNTOMA	QUÉ SIGNIFICA PROBABLEMENTE	CÓMO LO DETECTAS
El reward sube y la calidad retenida no	El modelo aprendió una señal interna que no generaliza.	Eval holdout y revisión de samples.
Las respuestas se vuelven más largas	El dataset premia detalle aunque no aporte.	<code>avg_completion_tokens</code> y coste por slice.
DPO mejora chosen accuracy pero rompe formato	La preferencia no incluía contrato de salida.	Validador JSON o tests de formato.
RLVR pasa tests visibles y falla casos nuevos	El grader no cubre suficiente variedad.	Casos ocultos y rotación de fixtures.
El modelo ajustado gana en media y cae en un área crítica	La media global oculta slices.	Métricas por familia de tarea.
El coste de inferencia sube más que la mejora	Se optimizó calidad sin restricción operativa.	Valor neto, latencia p95 y tokens.
El equipo no sabe explicar qué cambió	No hay card ni trazabilidad.	Falta de snapshot, config, eval y changelog.

El remedio no es "entrenar más". El remedio suele ser volver al contrato: mejores pares, mejor guía de anotación, mejor grader, eval retenida más dura o decisión de no usar RL en esa fase.

Herramientas y estado del arte

Fecha de corte: 8 de junio de 2026. El ecosistema se mueve rápido, así que las herramientas se citan por capacidad y no como receta cerrada.

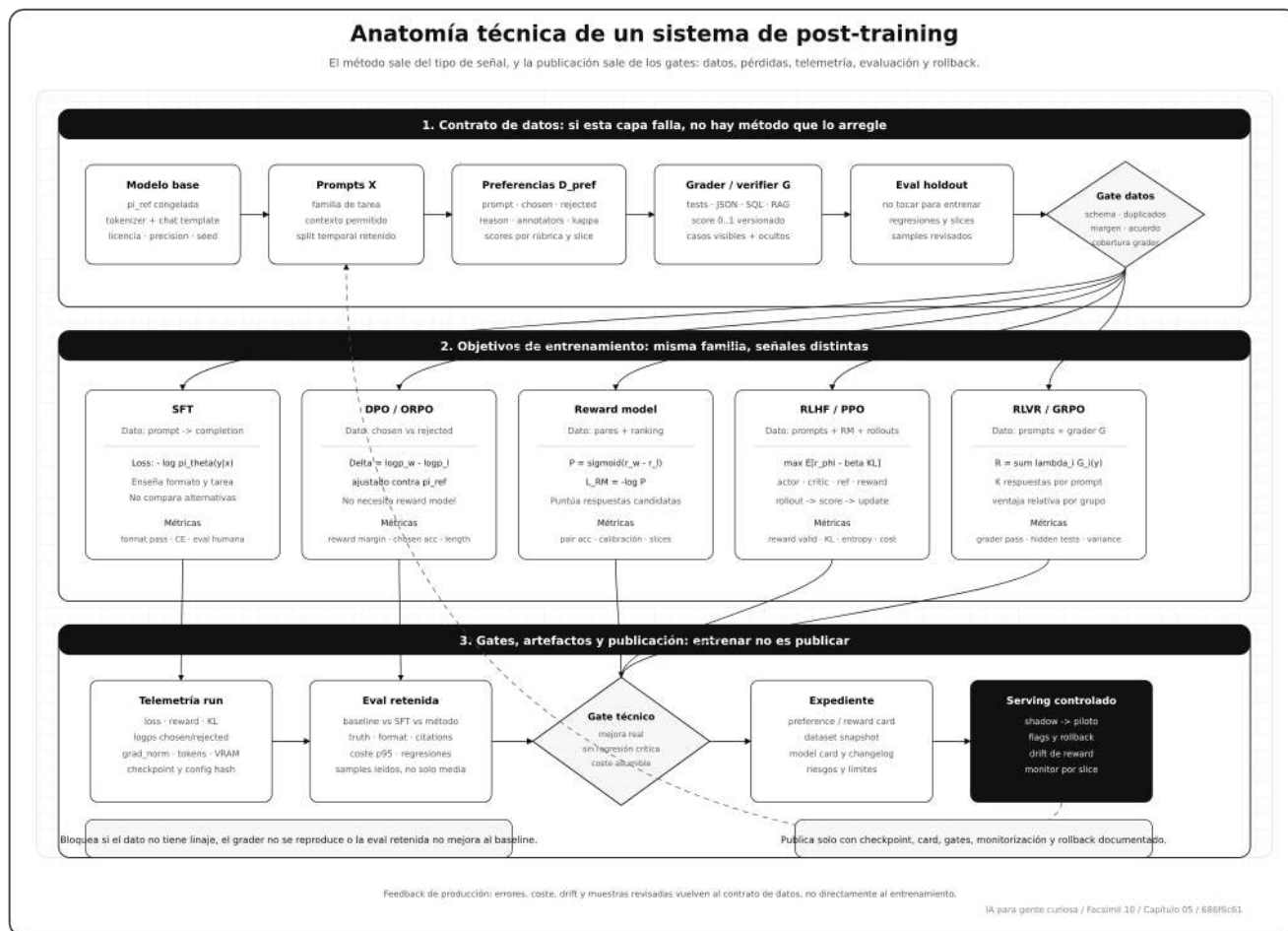
HERRAMIENTA	QUÉ APORTA	QUÉ REVISARÍA ANTES DE USARLA
Hugging Face TRL	Trainers para SFT, Reward, DPO, PPO, GRPO y variantes; formatos de dataset documentados.	Version exacta, formato <code>prompt/chosen/rejected</code> , métricas y soporte de PEFT.
OpenRLHF	Framework distribuido con PPO, GRPO, DPO, reward model, Ray, vLLM, checkpoints y opciones de alto rendimiento.	Complejidad operativa, GPUs, coherencia de rollouts y logprobs.
Axolotl	Configuración YAML para SFT, preference learning, GRPO, LoRA/QLoRA/full fine-tuning y mapas de hardware.	Que el método elegido encaje con tu señal real y tu VRAM.
Unsloth	Flujos rápidos para DPO, ORPO, KTO, GRPO y entrenamiento con menos VRAM en muchos escenarios.	Compatibilidad de modelos, kernels, cuantización y evaluación retenida.
OpenAI RFT/Graders	RFT con graders programables, métricas de reward y validación de graders.	Disponibilidad actual, coste, tipos de grader y errores de grading.

TRL documenta el formato esperado para DPO con `prompt` , `chosen` y `rejected` , además de tool calling y VLMs en datasets de preferencia.¹² OpenRLHF documenta algoritmos como PPO, REINFORCE++, GRPO y RLOO, motores híbridos con vLLM/Ray, rewards remotos y entrenamiento multi-turn.¹³ Axolotl ofrece una guía de decisión que separa SFT, preference learning, GRPO y reward modeling por dato requerido y coste de cómputo.¹⁴ Unsloth documenta flujos de DPO, ORPO y KTO integrados con TRL.¹⁵

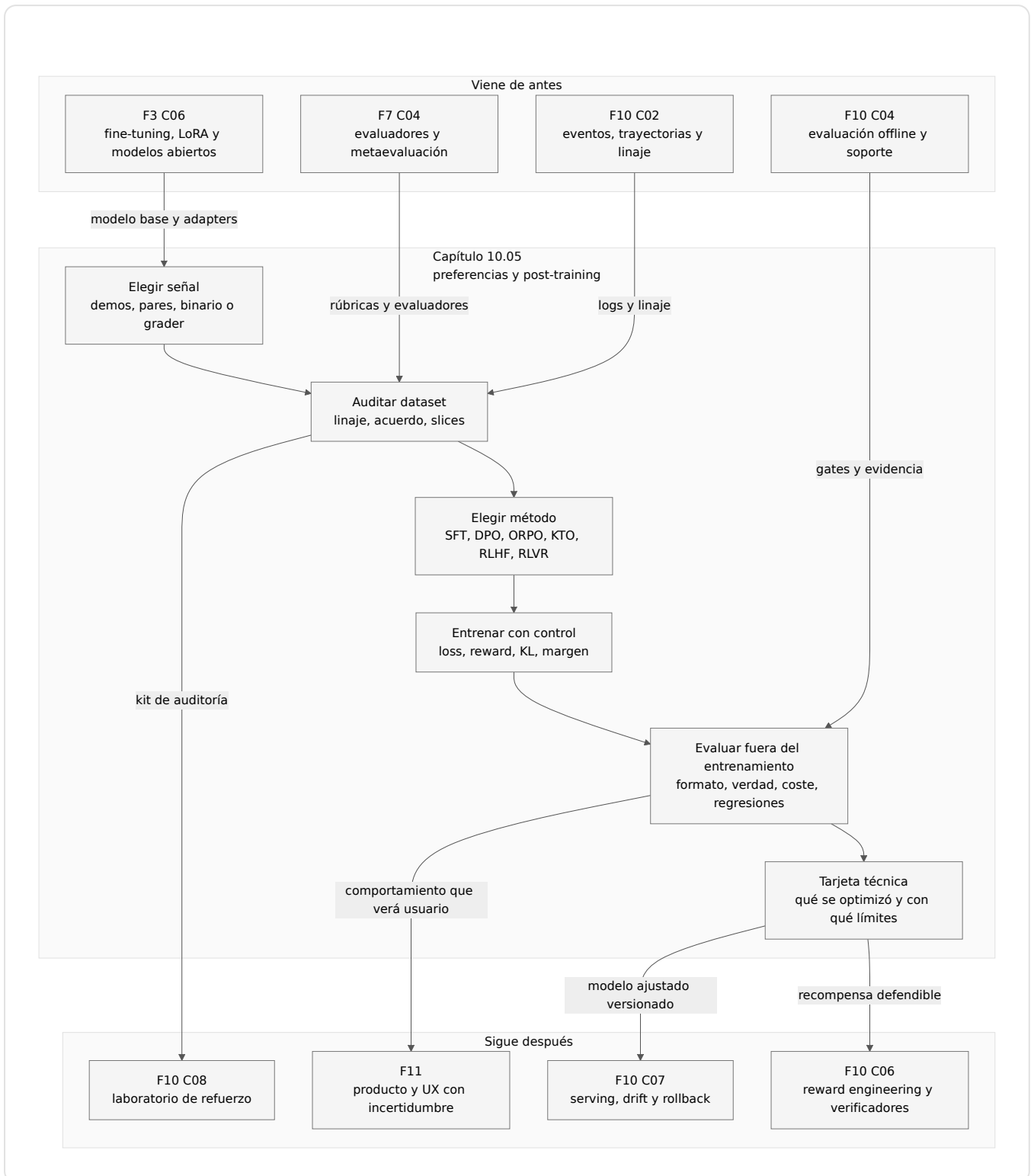
La pregunta profesional no es "¿qué herramienta está de moda?". Es esta:

PREGUNTA	SI LA RESPUESTA ES NO
¿Sé qué señal estoy optimizando?	No entrenes todavía.
¿Tengo dataset con linaje y partición retenida?	Primero ingeniería de datos.
¿Puedo reproducir el entrenamiento?	Versiona configs, seeds, modelo base y tokenizador.
¿Tengo métricas por slice?	La media global no basta.
¿Puedo comparar contra SFT y prompt baseline?	No sabes si RL aporta.
¿Sé cuánto cuesta servir el modelo ajustado?	El experimento no está listo para producto.

Anatomía técnica del post-training por señal



Cómo encaja todo



Vocabulario aprendido

TÉRMINO	QUÉ SIGNIFICA	CÓMO LO USARÍA
SFT	Ajuste supervisado con respuestas objetivo.	Para enseñar formato o tarea repetida.
Preference pair	Prompt con respuesta preferida y rechazada.	Base de DPO, ORPO y reward modeling.
Reward model	Modelo que puntúa respuestas.	Señal para RLHF o ranking.
Política de referencia	Modelo congelado usado como ancla.	Evita cambios excesivos.
KL	Distancia entre distribución entrenada y referencia.	Freno en RLHF/RFT.
DPO	Optimización directa de preferencias.	Si tengo pares limpios y quiero flujo offline.
KTO	Optimización desde señal binaria de deseabilidad.	Si el feedback no está pareado.
ORPO	Preferencia integrada con SFT por odds ratio.	Si quiero un flujo más compacto con pares.
RLAIF	Feedback de IA bajo principios.	Para escalar crítica, con auditoría humana de muestra.
RLVR	Refuerzo con recompensas verificables.	Código, matemáticas, SQL, graders.
GRPO	Optimización relativa por grupos.	Tareas con varias respuestas candidatas y reward.
Reward card	Documento que describe señal, datos, límites y gates.	Evidencia antes de publicar.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Decir "RLHF" cuando bastaba SFT	El nombre suena más avanzado.	Empezar por el tipo de dato disponible.
Creer que DPO arregla pares malos	La fórmula parece limpia.	Auditar duplicados, desacuerdo, margen y slices.
Mirar reward medio sin revisar samples	El número tranquiliza.	Leer respuestas retenidas y comparar con baseline.
Usar un grader sin versionarlo	Parece código auxiliar.	Tratarlo como parte del modelo: versión, tests y tarjeta.
Mezclar verdad, estilo y coste en una señal opaca	Todo se resume en una nota.	Separar componentes de reward y pesos.
No comparar contra prompt baseline y SFT	RL parece el siguiente paso natural.	Exigir mejora incremental medible.
Olvidar coste de serving	El entrenamiento pasa evals.	Medir tokens, latencia, VRAM y rollback antes de producto.

Antes de pasar página

- ¿Qué diferencia práctica hay entre SFT, DPO y RLHF?
- ¿Qué campos mínimos exigirías en un dataset de preferencias?
- ¿Por qué un reward model no es una autoridad?
- ¿Qué papel cumple la penalización KL?
- ¿Cuándo usarías KTO en vez de DPO?
- ¿Qué hace que RLVR sea atractivo y qué riesgo traslada al grader?
- ¿Qué tendría que contener una reward card antes de publicar un modelo ajustado?

Para saber más

Axolotl. (2026). *Which Fine-Tuning Method Should I Use?*
https://docs.axolotl.ai/docs/choosing_method.html

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., ... Kaplan, J. (2022). *Constitutional AI: Harmlessness from AI Feedback*. arXiv:2212.08073. <https://arxiv.org/abs/2212.08073>

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., ... Liang, W. (2025). *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. arXiv:2501.12948. <https://arxiv.org/abs/2501.12948>

Ethayarajh, K., Xu, W., Muennighoff, N., Jurafsky, D., & Kiela, D. (2024). *KTO: Model Alignment as Prospect Theoretic Optimization*. arXiv:2402.01306. <https://arxiv.org/abs/2402.01306>

Hong, J., Lee, N., & Thorne, J. (2024). *ORPO: Monolithic Preference Optimization without Reference Model*. arXiv:2403.07691. <https://arxiv.org/abs/2403.07691>

Hugging Face. (2026). *DPO Trainer*. https://huggingface.co/docs/trl/dpo_trainer

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., & Schulman, J. (2021). *WebGPT: Browser-assisted question-answering with human feedback*. arXiv:2112.09332. <https://arxiv.org/abs/2112.09332>

OpenAI. (2026). *Graders*. <https://developers.openai.com/api/docs/guides/graders>

OpenAI. (2026). *Reinforcement fine-tuning*. <https://developers.openai.com/api/docs/guides/reinforcement-fine-tuning>

OpenRLHF. (2026). *OpenRLHF documentation*. <https://openrlhf.readthedocs.io/en/latest/>

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). *Training language models to follow instructions with human feedback*. arXiv:2203.02155. <https://arxiv.org/abs/2203.02155>

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., & Finn, C. (2023). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. *Advances in Neural Information Processing Systems*, 36. <https://arxiv.org/abs/2305.18290>

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. arXiv:1707.06347. <https://arxiv.org/abs/1707.06347>

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D., & Christiano, P. F. (2020). *Learning to summarize from human feedback*. arXiv:2009.01325. <https://arxiv.org/abs/2009.01325>

Unsloth. (2026). *Reinforcement Learning - DPO, ORPO & KTO*. <https://docs.unsloth.ai/get-started/reinforcement-learning-rl-guide/reinforcement-learning-dpo-orpo-and-kto>

En resumen

Post-training es ingeniería de señal. SFT imita ejemplos; DPO y ORPO aprenden márgenes entre respuestas; KTO puede aprovechar feedback binario; RLHF usa reward model y optimización de política; RLAIIF escala crítica bajo principios; RLVR y RFT dependen de verificadores reproducibles. En todos los casos, el método no salva un dato mal diseñado.

El criterio profesional es sencillo de decir y difícil de cumplir: si no puedes explicar qué señal optimizaste, con qué dataset, qué evaluaciones pasaron, qué regresiones miraste y qué límites quedan, todavía no tienes un modelo ajustado defendible.

Notas

1. Ouyang, L. y otros (2022). *Training language models to follow instructions with human feedback*. arXiv:2203.02155. <https://arxiv.org/abs/2203.02155>. Consultado el 8 de junio de 2026. ↵
2. Stiennon, N. y otros (2020). *Learning to summarize from human feedback*. arXiv:2009.01325. <https://arxiv.org/abs/2009.01325>. Consultado el 8 de junio de 2026. ↵
3. Schulman, J. y otros (2017). *Proximal Policy Optimization Algorithms*. arXiv:1707.06347. <https://arxiv.org/abs/1707.06347>. Consultado el 8 de junio de 2026. ↵
4. OpenAI. (2026). *Reinforcement fine-tuning*. <https://developers.openai.com/api/docs/guides/reinforcement-fine-tuning>. Consultado el 8 de junio de 2026. ↵
5. Rafailov, R. y otros (2023). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv:2305.18290. <https://arxiv.org/abs/2305.18290>. Consultado el 8 de junio de 2026. ↵
6. Hugging Face. (2026). *DPO Trainer*. https://huggingface.co/docs/trl/dpo_trainer. Consultado el 8 de junio de 2026. ↵
7. Ethayarajh, K. y otros (2024). *KTO: Model Alignment as Prospect Theoretic Optimization*. arXiv:2402.01306. <https://arxiv.org/abs/2402.01306>. Consultado el 8 de junio de 2026. ↵
8. Hong, J. y otros (2024). *ORPO: Monolithic Preference Optimization without Reference Model*. arXiv:2403.07691. <https://arxiv.org/abs/2403.07691>. Consultado el 8 de junio de 2026. ↵

9. Bai, Y. y otros (2022). *Constitutional AI: Harmlessness from AI Feedback*. arXiv:2212.08073. <https://arxiv.org/abs/2212.08073>. Consultado el 8 de junio de 2026. ↵
- .0. DeepSeek-AI y otros (2025/2026). *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. arXiv:2501.12948. <https://arxiv.org/abs/2501.12948>. Consultado el 8 de junio de 2026. ↵
- .1. OpenAI. (2026). *Graders*. <https://developers.openai.com/api/docs/guides/graders>. Consultado el 8 de junio de 2026. ↵
- .2. Hugging Face. (2026). *DPO Trainer*. https://huggingface.co/docs/trl/dpo_trainer. Consultado el 8 de junio de 2026. ↵
- .3. OpenRLHF. (2026). *OpenRLHF documentation*. <https://openrlhf.readthedocs.io/en/latest/>. Consultado el 8 de junio de 2026. ↵
- .4. Axolotl. (2026). *Which Fine-Tuning Method Should I Use?* https://docs.axolotl.ai/docs/choosing_method.html. Consultado el 8 de junio de 2026. ↵
- .5. Unsloth. (2026). *Reinforcement Learning - DPO, ORPO & KTO*. <https://docs.unsloth.ai/get-started/reinforcement-learning-rl-guide/reinforcement-learning-dpo-orpo-and-kto>. Consultado el 8 de junio de 2026. ↵

Capítulo 06: Reward engineering: calidad de señal, verificadores y reward cards

La recompensa es una especificación bajo presión

En los capítulos anteriores hemos hablado de políticas, retorno, bandits, evaluación offline y post-training. Todo acaba chocando contra una pregunta incómoda: ¿qué está intentando maximizar el sistema?

La recompensa parece un número. En realidad es una especificación comprimida. Si esa especificación está mal escrita, la política no "entiende la intención"; optimiza lo que le dimos. En un sistema RAG, una reward puede premiar respuestas con cita. Bien. Pero si solo mira si aparece un identificador de documento, puede premiar citas decorativas. En un asistente de código, puede premiar tests que pasan. Bien. Pero si los tests son pobres, puede favorecer soluciones frágiles. En un agente con herramientas, puede premiar resolver rápido. Bien. Pero si no controla cambios de estado, puede sacrificar trazabilidad.

Sutton y Barto definen RL alrededor de señales de recompensa y retorno; esa idea es potente porque reduce una decisión secuencial a consecuencias medibles.¹ Silver, Singh, Precup y Sutton defendieron que maximizar recompensa puede explicar una gran variedad de comportamientos inteligentes, una tesis fuerte que conviene leer con cuidado en ingeniería aplicada.² En producto, la lección práctica no es "un número basta". Es justo la contraria: si el número manda, diseñarlo bien importa muchísimo.

Reward engineering es ese trabajo: convertir intención en una señal medible, versionada, auditable y resistente a cambios.

Qué no es una recompensa buena

Una recompensa buena no es la métrica más cómoda. `clicks`, `tokens`, `satisfacción`, `tiempo`, `respuestas largas`, `tests pasan`, `cita presente` o `like` pueden ser señales útiles, pero no son automáticamente el objetivo.

Tampoco es una suma de cosas bonitas:

`calidad + seguridad + rapidez + coste + satisfacción`

Si no defines cómo se mide cada término, qué peso tiene, qué casos debe pasar y cuándo bloquea, la fórmula es una frase. No una especificación.

Y no es estable para siempre. Una recompensa que funciona con un modelo base, un RAG, una política de herramientas y un tipo de usuario puede degradarse cuando cambia cualquiera de esas piezas.

La ley de Goodhart suele resumirse como el problema de convertir una medida en objetivo. Goodhart formuló esta preocupación en el contexto de gestión monetaria: cuando una regularidad observada se usa para control, puede dejar de comportarse igual.³ En IA aplicada, la lectura es directa: si una métrica se vuelve el objetivo del modelo, el modelo encontrará los bordes de esa métrica.

De intención a fórmula

Una recompensa compuesta es una suma ponderada de componentes: una escalarización lineal de varios objetivos en una sola señal. No es una receta universal; cada componente debe estar definido, medido y versionado:

$$R(x,y) = \sum_{j=1}^m w_j f_j(x,y)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
x	Entrada o contexto.	Pregunta del usuario + documentos recuperados.
y	Respuesta candidata.	Respuesta del asistente.
$f_j(x,y)$	Componente medible de la recompensa.	Exactitud, cita, formato, coste.
w_j	Peso del componente.	0,40 para exactitud.
m	Número de componentes.	7 términos.
$R(x,y)$	Recompensa total.	0,82

Para un asistente RAG, una instancia ilustrativa repartiría los pesos así. No son valores canónicos, sino una decisión de producto y riesgo que hay que medir, versionar y revisar:

TÉRMINO	PESO ILUSTRATIVO	QUÉ MIDE	CÓMO LO VERIFICARÍA
$R_{correcta}$	+0,40	La respuesta resuelve la tarea.	Revisión humana, grader de tarea o test.
$R_{evidencia}$	+0,22	La respuesta está soportada por fuente.	Verificador de cita y recuperación.
$R_{formato}$	+0,13	Cumple contrato de salida.	JSON schema, parser, tipos.
$R_{abstención}$	+0,12	Se abstiene si falta evidencia.	Clasificador de answerability o caso etiquetado.
$C_{latencia}$	-0,07	Tiempo de respuesta.	Traza p50/p95.
C_{tokens}	-0,04	Coste de generación.	Tokens de entrada/salida.
$C_{herramientas}$	-0,02	Coste y riesgo operativo.	Número y tipo de llamadas.

Los pesos no son verdad científica. Son una decisión de producto y riesgo. Precisamente por eso deben documentarse.

Antes de sumar: normalizar y separar lo no negociable

La fórmula anterior parece limpia, pero en ingeniería hay una trampa: no todas las señales nacen en la misma escala. La exactitud puede venir como 0 o 1. La latencia viene en milisegundos. Los tokens vienen como conteo. Una puntuación de evidencia puede salir de un verificador entre 0 y 1. Si sumas todo sin normalizar, el peso deja de significar lo que crees que significa.

La forma habitual de hacerlo es la normalización min-max, que lleva cada señal a un rango comparable:

$$\tilde{f}_j(x, y) = \frac{f_j(x, y) - f_j^{\min}}{f_j^{\max} - f_j^{\min} + \epsilon}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$f_j(x, y)$	Valor bruto de la señal.	1850 ms de latencia.
$f_j^{\min}$	Valor mínimo observado o pactado.	500 ms.
$f_j^{\max}$	Valor máximo observado o pactado.	5000 ms.
ϵ	Pequeño valor para evitar división por cero.	10^{-9} .
$\tilde{f}_j(x, y)$	Señal normalizada.	Valor comparable en 0..1.

Si menor es mejor, como latencia o coste, puedes convertirlo en penalización:

$$C_{\text{latencia}}(x, y) = \frac{\text{latencia}(x, y) - \text{latencia}_{\min}}{\text{latencia}_{\max} - \text{latencia}_{\min} + \epsilon}$$

Entonces un valor alto de C_{latencia} resta más. Si no haces esto, una mejora de 300 ms puede parecer enorme o irrelevante según el rango que haya caído en tus datos. Y si el rango se calcula con todo el tráfico mezclado, puedes castigar de forma injusta tareas naturalmente más lentas, como una consulta con herramienta frente a una respuesta directa. Por eso muchas reward cards normalizan por `slice` : RAG, SQL, salida estructurada, agente con herramientas, etc.

Hay una segunda trampa: no todo debe poder compensarse con puntos. Un JSON inválido no debería publicarse porque la respuesta sea muy bonita. Una cita que no soporta la frase no debería pasar porque el coste sea bajo. Una acción sobre un sistema externo no debería ejecutarse si falta aprobación humana o permiso técnico. Para eso usamos restricciones duras:

$$G(x, y) = 1[\text{JSON válido}] \cdot 1[\text{citas soportadas}] \cdot 1[\text{permiso correcto}]$$

Y la decisión de publicación se separa de la puntuación:

$$\text{publicar}(x, y) = 1[G(x, y) = 1] \cdot 1[R(x, y) \geq \tau]$$

ELEMENTO	QUÉ SIGNIFICA	EJEMPLO OPERATIVO
$G(x,y)$	Producto de restricciones duras.	Vale 0 si falla cualquier condición obligatoria.
$1[\cdot]$	Indicador: 1 si se cumple, 0 si no.	JSON válido: 1; JSON roto: 0.
τ	Umbral mínimo de reward.	Publicar solo si $R \geq 0,75$.
<code>slice</code>	Subconjunto con comportamiento propio.	<code>rag</code> , <code>sql</code> , <code>herramientas</code> , <code>privacidad</code> .

Este punto es muy de ingeniería: una reward no es solo una función matemática. Es una decisión de control. Tiene una parte continua, donde comparas candidatos, y una parte discreta, donde bloqueas salidas que no cumplen contrato.

Ejemplo numérico completo

Imagina un asistente RAG que responde una pregunta sobre una política interna. Tenemos dos respuestas candidatas. Las dos pasan el contrato mínimo: salida parseable, cita presente y ruta permitida. Ahora toca decidir cuál gana por recompensa.

COMPONENTE NORMALIZADO	CANDIDATO A	CANDIDATO B	LECTURA
<code>correctness</code>	0,80	1,00	B resuelve mejor la pregunta en abstracto.
<code>evidence</code>	1,00	0,35	A está mucho mejor soportado por documentos.
<code>format</code>	1,00	1,00	Ambos cumplen JSON schema.
<code>abstention</code>	0,00	0,00	En este caso sí hay fuente, no toca abstenerse.
<code>latency_cost</code>	0,45	0,12	B es más rápido.
<code>token_cost</code>	0,45	0,18	B usa menos tokens.
<code>tool_cost</code>	0,20	0,00	B usa menos herramientas.

Con los pesos declarados:

TÉRMINO	PESO	A	CONTRIBUCIÓN A	B	CONTRIBUCIÓN B
correctness	0,40	0,80	0,3200	1,00	0,4000
evidence	0,22	1,00	0,2200	0,35	0,0770
format	0,13	1,00	0,1300	1,00	0,1300
abstention	0,12	0,00	0,0000	0,00	0,0000
latency_cost	-0,07	0,45	-0,0315	0,12	-0,0084
token_cost	-0,04	0,45	-0,0180	0,18	-0,0072
tool_cost	-0,02	0,20	-0,0040	0,00	0,0000
Total			0,6165		0,5914

Gana A por 0,0251. No gana porque sea más barato ni porque sea más completo. Gana porque, en una tarea RAG, la evidencia pesa lo suficiente como para preferir una respuesta algo menos ambiciosa pero más defendible. Esta frase es importante: una reward card no solo produce un ganador, produce una explicación técnica del ganador.

Ahora reducimos el peso de `evidence` a la mitad, de 0,22 a 0,11. No cambiamos candidatos, solo criterio:

CANDIDATO	REWARD ORIGINAL	REWARD CON EVIDENCE A 0,11	QUÉ PASA
A	0,6165	0,5065	Pierde mucha puntuación porque dependía de evidencia fuerte.
B	0,5914	0,5529	Pierde poco porque tenía poca evidencia.

Con esa variación gana B. ¿Es malo? Depende. Si el equipo quería que el asistente priorizara respuestas más completas aunque la evidencia fuera parcial, el cambio es coherente. Si el equipo quería un RAG conservador para documentación interna, el cambio revela que el peso de evidencia no es un detalle: es una decisión de producto, riesgo y pedagogía.

Este es el tipo de ejemplo que conviene tener en la reward card. No para decorar. Para que cualquier persona del equipo pueda entender qué gana, qué pierde y qué criterio estamos imponiendo al sistema.

Tipos de señal y cuándo fiarte

SEÑAL	VENTAJA	RIESGO	USO RAZONABLE
Métrica de negocio	Conecta con resultado real.	Puede ser tardía y confusa.	Evaluación agregada y decisiones de producto.
Preferencia humana	Captura criterio experto.	Coste, desacuerdo y fatiga.	Pares, reward model, DPO/RLHF.
Verificador determinista	Reproducible y barato.	Mide solo lo programado.	JSON, SQL, tests, formato, citas.
Grader por modelo	Flexible para tareas abiertas.	Puede cambiar con modelo/prompts.	Rúbricas complejas con validación humana.
Proxy técnico	Fácil de medir.	Puede desplazar el objetivo real.	Coste, latencia, longitud, pasos.
Casos ocultos	Reduce ajuste al test visible.	Requiere mantenimiento.	Gate antes de publicar.

Christiano y otros mostraron que las preferencias humanas pueden definir recompensas para tareas donde escribir el reward manual es difícil, usando comparaciones de segmentos de comportamiento.⁴ Leike y otros formularon reward modeling como una dirección para escalar supervisión: aprender una función de recompensa de interacción con personas y optimizarla después.⁵

La advertencia: una recompensa aprendida sigue siendo una aproximación. No deja de necesitar validación.

Verificadores: programas que ponen límites

Un verificador convierte parte de la intención en una prueba reproducible. No resuelve todo, pero baja ambigüedad.

VERIFICADOR	ENTRADA	SALIDA	EJEMPLO
JSON schema	Texto generado.	pass/fail .	Campos obligatorios y tipos.
Test unitario	Código generado.	Tests pasados.	Función que valida eventos RL.
SQL fixture	Consulta SQL.	Filas esperadas.	Conteo de tickets por semana.
Verificador de cita	Respuesta + documentos.	Cita soportada o no.	ID de documento y fragmento.
Answerability	Pregunta + contexto.	Debe responder o abstenerse.	No hay fuente para una política.
Grader por modelo	Respuesta + rúbrica.	Puntuación 0..1.	Calidad de explicación técnica.

OpenAI documenta graders como evaluadores que devuelven una puntuación entre 0 y 1, incluyendo variantes como checks de texto, modelo, Python y combinaciones ponderadas.⁶ Su documentación de reinforcement fine-tuning conecta estos graders con métricas de entrenamiento y validación.⁷

La regla que usaría en un equipo: todo componente crítico de reward debe tener al menos una forma de verificación o revisión retenida. Si no, ese componente no debería decidir publicación por sí solo.

El verificador también se evalúa

Un grader o verificador no es verdad revelada. Es otro sistema que puede equivocarse. Si un verificador de citas marca como soportada una frase que el documento no sostiene, la reward premiará una mala respuesta. Si marca como no soportada una frase correcta, castigará respuestas buenas. En ambos casos la política aprende una señal deformada.

Por eso un verificador importante debe tener su propio conjunto de evaluación:

CASO	GOL D	PREDICCIÓN DEL VERIFICADOR	LECTURA
Cita correcta detectada como correcta	1	1	Verdadero positivo.
Cita incorrecta detectada como incorrecta	0	0	Verdadero negativo.
Cita incorrecta aceptada	0	1	Falso positivo.
Cita correcta rechazada	1	0	Falso negativo.

Con esa matriz calculamos métricas básicas:

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

MÉTRICA	QUÉ PREGUNTA RESPONDE	EN UNA REWARD CARD
Precisión	Cuando el verificador dice "pasa", ¿cuántas veces acierta?	Importa si el verificador desbloquea publicación.
Recall	De los casos que deberían pasar, ¿cuántos detecta?	Importa si no quieres castigar respuestas buenas.
Accuracy	¿Qué proporción total clasifica bien?	Útil, pero puede engañar si las clases están desbalanceadas.
Falsos positivos	¿Qué se cuela como válido sin serlo?	Puede elevar la reward de respuestas incorrectas.
Falsos negativos	¿Qué se castiga siendo válido?	Puede empujar al modelo hacia respuestas conservadoras o pobres.

Ejemplo: si un verificador de cita tiene mucha precisión y poco recall, suele ser estricto. Puede venir bien en un dominio regulado, pero quizá castigue demasiadas respuestas correctas. Si tiene mucho recall y poca precisión, deja pasar demasiadas citas débiles. En un RAG interno, eso puede convertir una reward de evidencia en una reward de "poner IDs de documentos".

La decisión no es elegir una métrica bonita. Es decidir qué error pesa más para el producto, declarar esa elección y volver a medir cuando cambien documentos, plantilla de respuesta, modelo base o política de recuperación.

Reward card: el expediente de la señal

Una reward card es una ficha técnica de la recompensa. No es un documento bonito para enseñar al final. Es el contrato que evita que el equipo olvide qué se está optimizando.

CAMPO	PREGUNTA QUE RESPONDE
Objetivo	¿Qué comportamiento queremos mejorar?
No objetivos	¿Qué no debe premiarse aunque mejore la métrica?
Términos	¿Qué componentes tiene la recompensa?
Pesos	¿Cuánto pesa cada componente y por qué?
Verificadores	¿Cómo se mide cada componente?
Casos visibles	¿Qué ejemplos usamos para depurar?
Casos ocultos	¿Qué ejemplos protegemos para gate?
Slices	¿Dónde puede fallar aunque la media pase?
Coste	¿Qué penalizamos por tokens, latencia y herramientas?
Límites	¿Qué no cubre esta reward card?
Criterio de cambio	¿Cuándo hay que versionarla?

La reward card debe vivir cerca del código y de los datos. Si está en una presentación suelta, se perderá justo cuando más haga falta.

Casos negativos y ocultos

El reward no se prueba solo con casos donde la respuesta correcta gana de forma obvia. Necesita casos que fuercen bordes:

CASO	QUÉ COMPRUEBA
Respuesta correcta sin cita	Que evidencia importa.
Cita presente pero no soporta la frase	Que la cita no sea decorativa.
JSON con buen contenido pero inválido	Que formato importa.
Pregunta sin fuente	Que abstención importa.
Respuesta larga y elegante pero costosa	Que longitud no se premie sola.
Tool timeout	Que el sistema no rellene resultados no observados.
Slice crítico pequeño	Que la media global no oculte caída.

Amodei y otros colocaron el problema de objetivo incorrecto y optimización literal de especificaciones dentro de los problemas concretos de seguridad en sistemas de aprendizaje.⁸ Krakovna y otros recopilaron ejemplos de sistemas que cumplen la especificación literal sin cumplir la intención práctica, lo que ilustra por qué necesitamos casos negativos y verificadores.⁹

No se trata de asustar al alumno. Se trata de enseñar una disciplina: cuando una señal manda, hay que probar sus bordes.

Ejemplos de reward por dominio

La reward cambia según el sistema. No existe una plantilla universal, pero sí una manera de pensar: objetivo, señales, restricciones duras, casos retenidos y monitorización.

SISTEMA	QUÉ OPTIMIZA	SEÑALES ÚTILES	RESTRICCIÓN DURA TÍPICA	CASO QUE PONDRÍA SÍ O SÍ
RAG documental	Responder con evidencia real.	Correctitud, soporte de cita, abstención, coste.	No afirmar si el documento recuperado no soporta la frase.	Pregunta sin fuente suficiente.
SQL assistant	Generar consulta ejecutable y correcta.	Resultado esperado, sintaxis, coste de query, explicación.	La consulta debe ejecutarse en fixture controlado.	Query correcta en una tabla, incorrecta en otra por join mal hecho.
Agente con herramientas	Resolver una tarea con trazabilidad.	Éxito de tarea, pasos, coste, permisos, recuperación ante error.	No hacer cambios de estado sin aprobación o permiso técnico.	Tool timeout con respuesta honesta.
Asistente de código	Producir código mantenible.	Tests, tipos, lint, complejidad, cobertura de casos.	No pasar si rompe un test existente.	Solución que pasa un test simple pero falla borde.
Salida estructurada	Devolver contrato validable.	JSON schema, campos obligatorios, tipos, catálogo.	El parser debe cargar la salida sin reparación manual.	Buen contenido con JSON inválido.
Routing de modelos	Elegir modelo adecuado por caso.	Calidad, coste, latencia, confianza, fallback.	No enviar datos sensibles a una ruta no permitida.	Caso barato que no necesita modelo caro.

Fíjate en que la reward no sustituye al contrato del sistema. Lo complementa. En salida estructurada, el contrato parseable va primero. En herramientas, permisos y trazabilidad van primero. En RAG, evidencia y abstención van primero. La puntuación sirve para comparar candidatos dentro de un marco válido, no para borrar errores de diseño.

Sensibilidad de pesos

Una reward card debe incluir una pregunta incómoda: ¿qué pasa si cambio un poco los pesos?

Si el candidato ganador cambia al mover `evidence` de 0,22 a 0,20, la señal es frágil. Si al duplicar una penalización de coste siguen ganando los mismos candidatos correctos, quizá la reward es estable. Este análisis no demuestra que la reward sea buena, pero detecta fórmulas que dependen demasiado de una decisión arbitraria.

Formalmente puedes comparar ganadores:

$$\Delta_{\text{winners}}(w_j, \alpha) = \sum_{i=1}^n 1 \left[\operatorname{argmax}_y R_i(y; w_j) \neq \operatorname{argmax}_y R_i(y; \alpha w_j) \right]$$

SÍMBOLO	SIGNIFICADO	LECTURA
w_j	Peso original del término j .	Peso de evidencia, formato o coste.
α	Multiplicador aplicado al peso.	0,50; 0,75; 1,25; 1,50.
n	Número de casos evaluados.	Casos visibles y ocultos.
Δ_{winners}	Número de ganadores que cambian.	Mayor valor implica mayor sensibilidad.

La señal que quieres para producción no es "nunca cambia". Hay decisiones donde coste o evidencia deben cambiar el ganador. Lo que buscas es que los cambios sean explicables. Si al variar `token_cost` cambia un caso de privacidad, hay que mirar si estás mezclando señales sin control. Si al variar `evidence` cambia un caso RAG dudoso, puede ser justo lo que esperabas.

Calibrar el umbral

El umbral τ no se elige mirando una respuesta suelta. Se calibra contra un conjunto retenido donde sabes qué salidas deberían pasar y cuáles deberían bloquearse. Si τ es demasiado bajo, pasan respuestas débiles. Si τ es demasiado alto, bloqueas respuestas útiles y el sistema se vuelve torpe.

Para cada candidato retenido calculas:

$$\hat{p}(y) = 1[R(x, y) \geq \tau]$$

y lo comparas con una etiqueta de referencia:

$$z(y) \in \{0, 1\}$$

SÍMBOLO	QUÉ SIGNIFICA	EJEMPLO
τ	Umbral de publicación o selección.	0,75 en un asistente documental.
$\hat{p}(y)$	Decisión del sistema con ese umbral.	1 si pasa, 0 si bloquea.
$z(y)$	Etiqueta retenida.	1 si el equipo acepta la salida.
Falso pase	$\hat{p} = 1, z = 0.$	Sale una respuesta no aceptable.
Falso bloqueo	$\hat{p} = 0, z = 1.$	Se descarta una respuesta útil.

Una tabla mínima de calibración podría verse así:

UMBRAL T	PASS RATE	FALSOS PASES	FALSOS BLOQUEOS	LECTURA
0,55	0,91	7	1	Demasiado permisivo para RAG interno.
0,65	0,78	3	3	Equilibrado si el coste de revisar es bajo.
0,75	0,62	1	7	Más estricto; útil si evidencia importa mucho.
0,85	0,34	0	18	Quizá demasiado conservador.

No hay un único umbral correcto. Puede haber un τ_{rag} , un τ_{sql} , un $\tau_{\text{herramientas}}$ y un $\tau_{\text{privacidad}}$. En SQL, un falso pase puede romper una ejecución; en un resumen interno, un falso bloqueo quizá solo obliga a revisar manualmente. El umbral debe reflejar ese coste.

La regla práctica: calibra por slice, guarda la curva de decisión y revisa ejemplos concretos, no solo la media. Una reward con buen promedio puede tener un umbral pésimo para un segmento pequeño.

Producción: reward drift y monitorización

La reward card no termina cuando se publica un experimento. En serving, el sistema cambia: nuevos documentos, nuevas consultas, nuevas plantillas, nuevos modelos, cambios de latencia y modificaciones de herramientas. En ese punto hay que monitorizar la relación entre reward y resultado real.

SEÑAL EN PRODUCCIÓN	QUÉ PUEDE INDICAR	QUÉ HARÍA
<code>reward_mean</code> sube y satisfacción baja	La política aprendió a complacer la métrica.	Revisar casos negativos y ejemplos retenidos.
<code>evidence_pass_rate</code> sube demasiado rápido	El verificador puede estar aceptando señales débiles.	Auditar muestras y matriz del verificador.
<code>abstention_rate</code> cae	El sistema responde aunque falte fuente.	Revisar answerability y casos sin evidencia.
<code>token_cost_p95</code> sube	La reward no penaliza suficiente longitud o herramientas.	Revisar pesos y normalización por slice.
<code>case_pass_rate</code> pasa en global y falla un slice	La media oculta un segmento crítico.	Gate por slice, no solo por media.
<code>grader_precision</code> cambia	El evaluador dejó de comportarse igual.	Recalibrar grader o congelar versión.

Para un equipo de ingeniería, la reward card debería versionarse junto a cuatro cosas: dataset de evaluación, verificadores, prompts o contratos de salida, y política candidata. Si solo versionas la fórmula, no podrás explicar por qué cambió una decisión.

Qué se guarda en cada run

La reward card no sirve de mucho si producción no guarda las trazas necesarias. Cuando una respuesta gana, el equipo debería poder reconstruir por qué ganó: qué política la generó, qué reward card se usó, qué verificadores puntuaron, qué gates pasaron y qué valores tuvo cada componente.

Una traza mínima para este capítulo tendría esta forma:

```
{
  "trace_id": "f10-c06-0001",
  "run_at": "2026-06-09T17:30:00Z",
  "policy_version": "rag-policy-2026-06-09",
  "reward_card_version": "1.0.0",
  "input_slice": "rag",
  "candidate_id": "a",
  "component_scores": {
    "correctness": 0.8,
    "evidence": 1.0,
    "format": 1.0,
    "abstention": 0.0,
    "latency_cost": 0.45,
    "token_cost": 0.45,
    "tool_cost": 0.2
  },
  "hard_gates": {
    "valid_output_contract": true,
    "supported_claims": true,
    "answerability_or_abstention": true
  },
  "reward": 0.6165,
  "winner": true,
  "latency_ms": 1850,
  "input_tokens": 1420,
  "output_tokens": 310,
  "grader_versions": {
    "citation_support_v1": "2026-06-09",
    "json_schema_v1": "1.0.0",
    "answerability_v1": "2026-06-09"
  },
  "dataset_version": "reward-eval-2026-06-09"
}
```

CAMPO	POR QUÉ IMPORTA
<code>trace_id</code>	Une logs, respuesta, evaluación y revisión posterior.
<code>policy_version</code>	Permite saber qué política o modelo produjo la salida.
<code>reward_card_version</code>	Evita comparar puntuaciones de fórmulas distintas como si fueran iguales.
<code>input_slice</code>	Permite detectar drift por segmento.
<code>component_scores</code>	Explica el reward total y permite auditoría por término.
<code>hard_gates</code>	Separa condiciones obligatorias de puntuación ponderada.
<code>grader_versions</code>	Hace reproducible la evaluación si cambia un grader.
<code>dataset_version</code>	Conecta la decisión con el conjunto de casos retenidos.

Este contrato, `reward_run_trace_contract.json`, no es un estándar universal; es una propuesta mínima para que el alumno entienda qué debe guardar si quiere operar una reward en serio.

Criterios para pasar al siguiente paso

Una reward card no debería pasar de documento a experimento porque "parece razonable". Debería pasar por una checklist.

CRITERIO	MÍNIMO RAZONABLE	QUÉ MIRARÍA
Casos	Al menos 8-10 iniciales, creciendo con el producto.	No solo casos felices.
Slices	Varios segmentos reales.	RAG, SQL, coste, herramientas, privacidad.
Casos ocultos	25 % o más en una primera práctica.	No ajustar todo contra lo visible.
Pass rate	90 % o más antes de experimento controlado.	Mirar también errores concretos.
Restricciones duras	Todas con verificador real.	Ninguna condición obligatoria en <code>none</code> .
Normalización	Costes normalizados antes de sumar.	Latencia y tokens por slice.
Verificador	Matriz de confusión revisada.	Falsos pases y falsos bloqueos.
Sensibilidad	Cambios de ganador explicables.	Identificar casos como <code>sensibilidad_evidencia</code> .
Trazas	Campos mínimos definidos.	Versiones de policy, reward, dataset y graders.
CI	Falla si <code>status=block</code> .	No depender de lectura manual.

Esta tabla no sustituye al criterio del equipo. Lo fuerza a hacerse explícito. Y eso, en sistemas de IA, ya es media batalla ganada.

Qué se lleva un equipo de ingeniería

Al terminar este capítulo, un equipo debería poder llevarse tres piezas reutilizables.

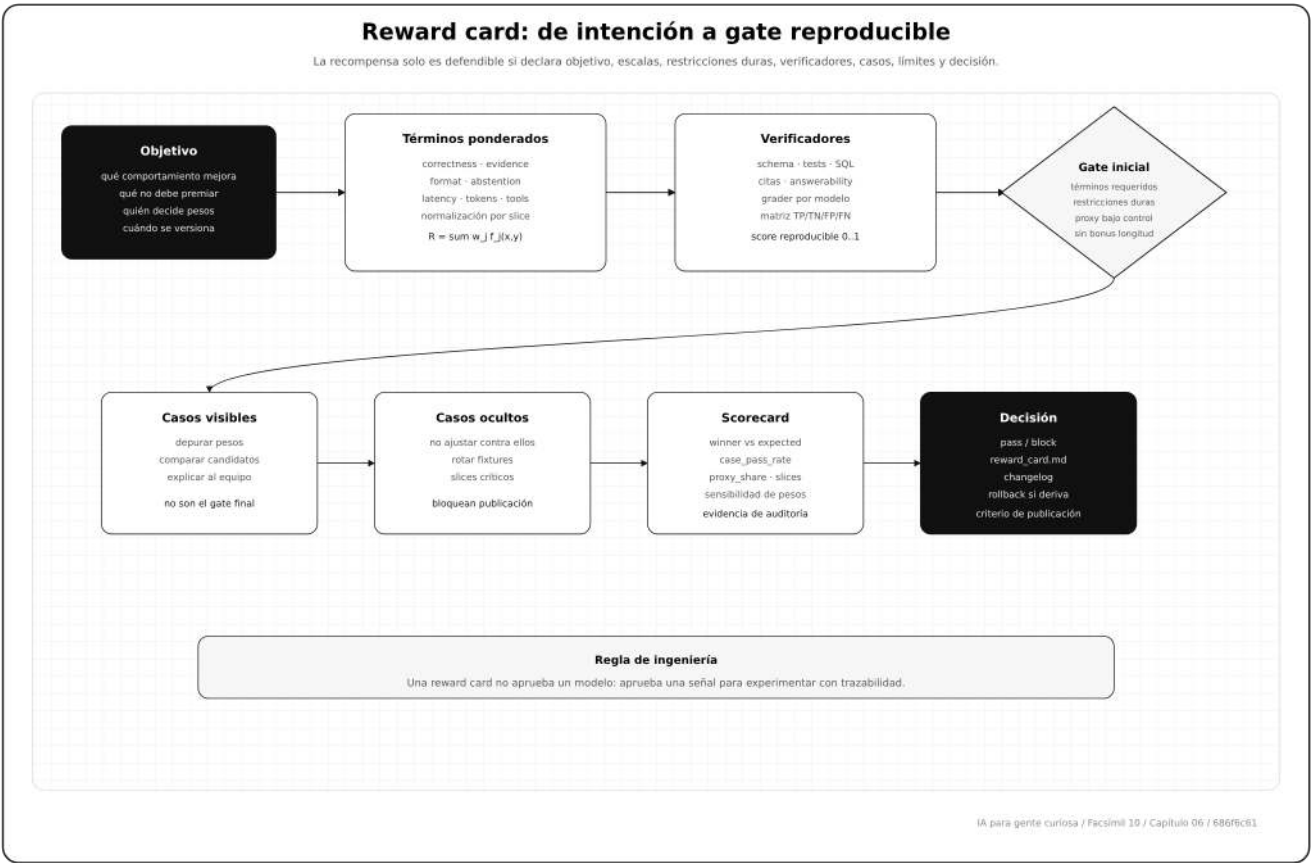
PIEZA	PARA QUÉ SIRVE	CUÁNDO SE USA
Validador de trazas	Comprueba que una run guarda campos, versiones, gates y reward coherente.	Antes de confiar en métricas de producción.
Calibrador de umbral	Calcula falsos pases y falsos bloqueos para varios τ .	Antes de seleccionar o publicar candidatos.
Plantilla de PR	Obliga a justificar cambios de objetivo, pesos, casos, sensibilidad y operación.	Cada vez que cambia una reward card.

El validador de trazas evita una situación muy común: tener un dashboard bonito sin poder reconstruir una decisión. Si falta `reward_card_version` , no sabes qué fórmula puntuó. Si falta `grader_versions` , no sabes si la evaluación cambió. Si el `reward` guardado no coincide con los componentes, no sabes si hay un bug de cálculo o una transformación no documentada.

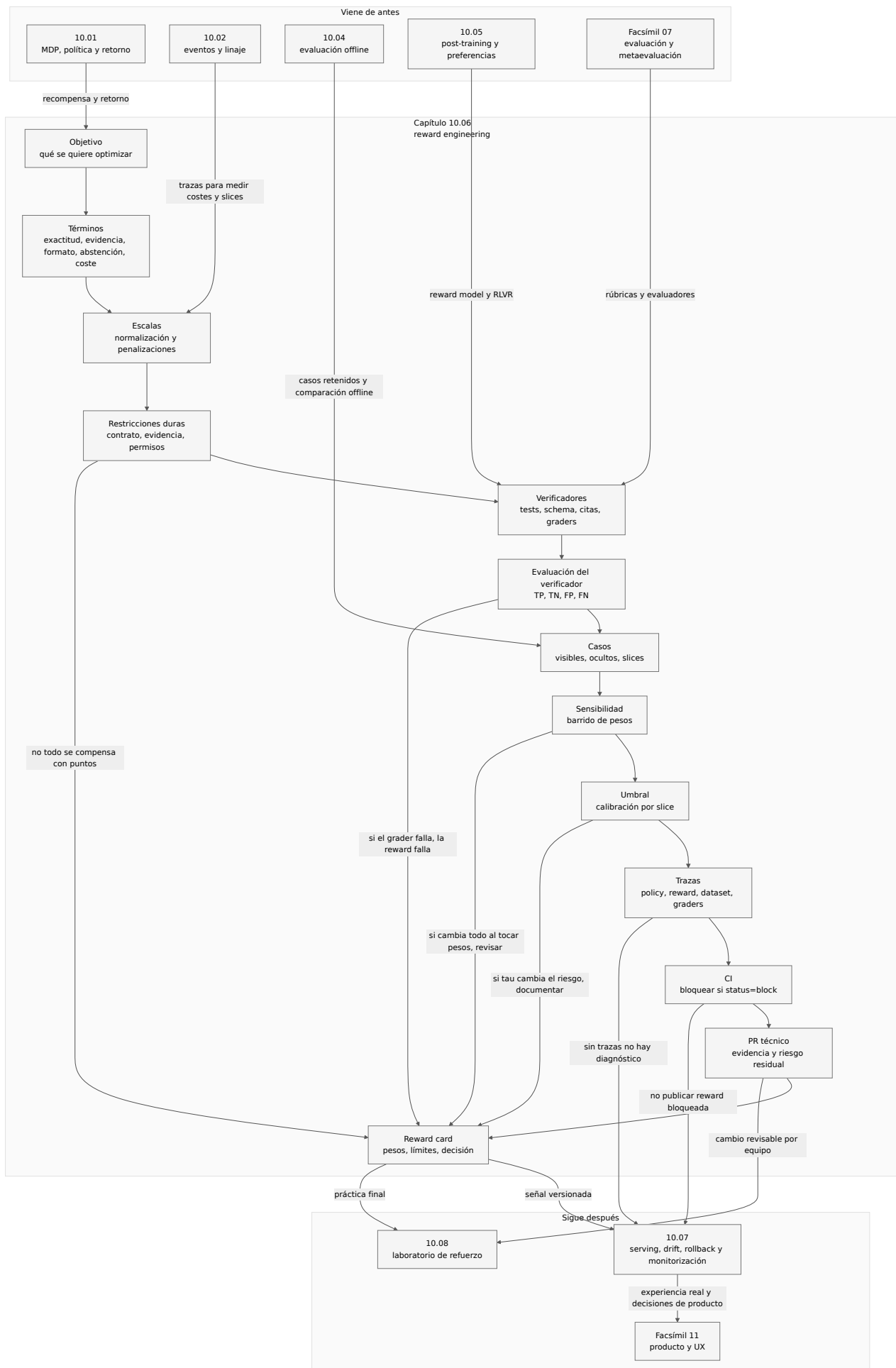
El calibrador de umbral convierte una discusión vaga en una tabla revisable. En lugar de decir "0,75 parece bien", el equipo mira falsos pases, falsos bloqueos y casos concretos por slice. Para RAG interno quizá priorizas no dejar pasar respuestas sin soporte. Para SQL quizá priorizas no ejecutar una consulta incorrecta. Para un asistente de redacción quizá aceptas más falsos bloqueos si la revisión humana es barata.

La plantilla de PR es la disciplina que evita cambios silenciosos. Cambiar `evidence` de 0,22 a 0,11 no es tocar un número: cambia qué tipo de respuesta gana. Un PR de reward card debe enseñar `reward_card_decision.md` , `sensitivity_report.csv` , `threshold_recommendation.md` , matriz de verificador y trazas. Si no, el equipo está aprobando una intuición, no una especificación.

Anatomía de una reward card defendible



Cómo encaja todo



Vocabulario aprendido

TÉRMINO	QUÉ SIGNIFICA	CÓMO LO USARÍA
Reward	Señal que una política maximiza.	Resultado numérico de una especificación.
Proxy	Medida indirecta del objetivo.	Latencia, longitud, clicks, coste.
Verificador	Prueba reproducible de un componente.	JSON schema, test, SQL, cita.
Grader	Evaluador que devuelve puntuación.	Puede ser código, modelo o mezcla.
Caso oculto	Caso reservado para gate.	No se usa para ajustar pesos.
Reward card	Expediente de la señal.	Documento versionado con pesos y límites.
Reward drift	Pérdida de relación entre reward y objetivo real.	Cambios de datos, producto o usuarios.
Normalización	Conversión de señales a una escala comparable.	Evita sumar milisegundos con exactitud como si fueran iguales.
Restricción dura	Condición que bloquea aunque la puntuación sea alta.	JSON válido, cita soportada, permiso técnico.
Matriz de confusión	TP, TN, FP y FN de un verificador.	Mide si el grader merece confianza operativa.
Sensibilidad de pesos	Cambio de decisiones al variar pesos.	Detecta rewards frágiles.
Umbral	Valor mínimo para pasar una decisión.	Calibrar τ por slice antes de publicar.
Traza de reward	Registro completo de una run evaluada.	Reconstruir por qué ganó una respuesta.
Calibración de umbral	Barrido de thresholds contra casos etiquetados.	Elegir τ mirando falsos pases y bloqueos.
Plantilla de PR	Contrato de revisión para cambios de reward.	Evitar cambios de pesos sin evidencia.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Premiar lo fácil de medir	La métrica está a mano.	Separar objetivo real, proxy y coste.
No poner casos ocultos	Da pereza mantenerlos.	Reservarlos como gate de publicación.
Usar un grader sin probarlo	Devuelve un número.	Auditarlo con casos donde sabemos la respuesta.
Mezclar coste y calidad sin pesos claros	Todo se mete en una fórmula.	Declarar pesos y revisar sensibilidad.
No normalizar escalas	Latencia, tokens y calidad viven en rangos distintos.	Normalizar por slice antes de sumar.
Compensar condiciones obligatorias con puntos	Una respuesta inválida puede ganar por otros términos.	Separar restricciones duras de reward ponderada.
Premiar longitud	Parece que una respuesta larga trabaja más.	Medir tokens y no dar bonus por extensión.
Olvidar slices pequeños	La media pasa.	Bloquear si cae un slice crítico.
Elegir τ a ojo	El umbral parece un número menor.	Calibrarlo con casos retenidos y coste de error.
No guardar trazas	En producción solo queda el resultado final.	Registrar policy, reward card, dataset, graders y componentes.
Cambiar una reward sin PR técnico	Parece un ajuste local.	Exigir sensibilidad, calibración, trazas y riesgo residual.
No versionar la reward card	Parece documentación.	Versionarla como código y datos.

Antes de pasar página

- ¿Por qué una recompensa es una especificación bajo presión?
- ¿Qué diferencia hay entre objetivo real y proxy?
- ¿Qué términos pondrías en una reward card para un asistente RAG?
- ¿Cuándo un verificador determinista es mejor que un grader por modelo?
- ¿Por qué un verificador necesita su propia matriz de confusión?

- ¿Qué diferencia hay entre una penalización ponderada y una restricción dura?
- ¿Qué significa que una reward sea sensible a pesos?
- ¿Cómo elegirías un umbral para RAG y otro para SQL?
- ¿Qué campos mínimos guardarías en una traza de reward?
- ¿Qué debería fallar en CI cuando una reward card queda en `block`?
- ¿Qué evidencia pedirías en un PR que cambia pesos?
- ¿Por qué hacen falta casos ocultos?
- ¿Qué bloquearías antes de publicar una política?

Para saber más

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). *Concrete Problems in AI Safety*. arXiv:1606.06565. <https://arxiv.org/abs/1606.06565>

Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). *Deep Reinforcement Learning from Human Preferences*. arXiv:1706.03741. <https://arxiv.org/abs/1706.03741>

Goodhart, C. A. E. (1975). *Problems of Monetary Management: The U.K. Experience*. Reserve Bank of Australia. <https://www.rba.gov.au/publications/confs/1975/>

Krakovna, V., Uesato, J., Mikulik, V., Rahtz, M., Everitt, T., Kumar, R., Kenton, Z., Leike, J., & Legg, S. (2020). *Specification Gaming: The Flip Side of AI Ingenuity*. Google DeepMind. <https://deepmind.google/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity>

Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., & Legg, S. (2018). *Scalable Agent Alignment via Reward Modeling: A Research Direction*. arXiv:1811.07871. <https://arxiv.org/abs/1811.07871>

OpenAI. (2026). *Graders*. <https://platform.openai.com/docs/guides/graders/>

OpenAI. (2026). *Reinforcement Fine-Tuning*. <https://developers.openai.com/api/docs/guides/reinforcement-fine-tuning>

Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299, 103535. <https://doi.org/10.1016/j.artint.2021.103535>

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press. <http://incompleteideas.net/book/the-book-2nd.html>

En resumen

Reward engineering es diseñar la señal que una política va a obedecer. Si la señal premia una aproximación pobre, la política no falla por falta de inteligencia: falla porque le dimos un objetivo incompleto.

Una reward card defendible declara objetivo, términos, normalización, restricciones duras, pesos, verificadores, evaluación del verificador, casos visibles, casos ocultos, slices, umbrales, límites, trazas, CI y decisión. No garantiza que el modelo sea perfecto, pero permite experimentar sin perder el rastro de qué se está optimizando, qué se bloquea y por qué.

Notas

1. Sutton, R. S. y Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2.ª ed.). MIT Press. <http://incompleteideas.net/book/the-book-2nd.html>. Consultado el 8 de junio de 2026. ↵
2. Silver, D., Singh, S., Precup, D. y Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299, 103535. <https://doi.org/10.1016/j.artint.2021.103535> ↵
3. Goodhart, C. A. E. (1975). *Problems of Monetary Management: The U.K. Experience*. Reserve Bank of Australia. <https://www.rba.gov.au/publications/confs/1975/>. Consultado el 8 de junio de 2026. ↵
4. Christiano, P. F. y otros (2017). *Deep Reinforcement Learning from Human Preferences*. arXiv:1706.03741. <https://arxiv.org/abs/1706.03741>. Consultado el 8 de junio de 2026. ↵
5. Leike, J. y otros (2018). *Scalable Agent Alignment via Reward Modeling: A Research Direction*. arXiv:1811.07871. <https://arxiv.org/abs/1811.07871>. Consultado el 8 de junio de 2026. ↵
6. OpenAI. (2026). *Graders*. <https://platform.openai.com/docs/guides/graders/>. Consultado el 8 de junio de 2026. ↵
7. OpenAI. (2026). *Reinforcement fine-tuning*. <https://developers.openai.com/api/docs/guides/reinforcement-fine-tuning>. Consultado el 8 de junio de 2026. ↵
8. Amodei, D. y otros (2016). *Concrete Problems in AI Safety*. arXiv:1606.06565. <https://arxiv.org/abs/1606.06565>. Consultado el 8 de junio de 2026. ↵
9. Krakovna, V. y otros (2020). *Specification Gaming: The Flip Side of AI Ingenuity*. Google DeepMind. <https://deepmind.google/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity>. Consultado el 8 de junio de 2026. ↵

Capítulo 07: Serving de políticas: monitorización, drift y cambios controlados

La política no termina cuando pasa evaluación

Una política puede pasar evaluación offline, tener una reward card razonable y aun así comportarse mal al vivir en producción. No porque el algoritmo sea misterioso. Porque producción no es el notebook. Producción trae tráfico cambiante, latencia, herramientas que fallan, usuarios con patrones nuevos, documentos actualizados, costes, trazas incompletas y decisiones que hay que poder deshacer.

En este capítulo dejamos de mirar la política como una fórmula aislada y la miramos como un componente operativo. Una política servida no es solo $\pi(a | x)$. Es $\pi(a | x)$ más versión, traza, SLO, gate, feature flag, política de reserva, ventana de referencia, monitorización por slice y runbook.

Fecha de corte: 9 de junio de 2026. Para la parte de herramientas y operación hemos revisado documentación oficial de OpenTelemetry, LaunchDarkly y Evidently, y la conectamos con literatura estable sobre deuda técnica de ML, producción de modelos, SRE y drift.¹²³

La idea central:

Una política no se publica: se sirve, se mide, se limita y se puede retirar.

Qué no es servir una política

Servir una política no es desplegar un endpoint y mirar si responde. Eso solo comprueba que hay una ruta técnica. No comprueba si la política está tomando buenas decisiones.

Tampoco es activar una feature flag sin contrato. Una flag controla exposición; la política decide acciones. Si mezclas ambas cosas, acabas sin saber si el problema viene del rollout, del modelo, de la recompensa, del contexto, de la herramienta o de la población que recibió la variante.

Y no es monitorizar una media global. Si `reward_mean` sube pero `privacidad` baja, el sistema no está sano. Si baja la latencia porque deja de llamar a una herramienta necesaria, tampoco. Si mejora la aceptación inmediata pero aumenta fallback, quizá has movido el coste a soporte, revisión o reintentos.

Sculley y otros describieron que los sistemas de ML acumulan deuda técnica por dependencias ocultas, configuración, bucles de feedback y cambios del mundo externo.⁴ Breck y otros propusieron el ML Test Score como una rúbrica de preparación productiva donde las pruebas y la monitorización son parte del sistema, no un adorno final.⁵

La lección práctica para este capítulo: una política que no deja evidencia operativa no está lista para producción.

Qué sí es el serving de políticas

Serving de políticas es la capa que recibe una petición, construye el contexto, elige una acción según una política versionada, ejecuta o propone esa acción, registra la decisión y aplica gates antes de aumentar exposición.

En un sistema de IA puede aparecer en muchos sitios:

SISTEMA	POLÍTICA	ACCIÓN
Routing de modelos	Elegir modelo según tarea y coste.	<code>small_rag_model</code> , <code>large_reasoning_model</code> , <code>local_model</code> .
RAG	Elegir estrategia de recuperación.	<code>top_k=4</code> , <code>top_k=12</code> , reranker, abstención.
Agente con herramientas	Elegir siguiente herramienta o parada.	Buscar, consultar base de datos, pedir aprobación.
Post-training	Elegir variante candidata.	Política estable o política ajustada.
Producto	Elegir experiencia controlada.	Mostrar respuesta directa, pedir aclaración, derivar a humano.

El serving correcto separa cinco planos:

PLANO	PREGUNTA
Política	¿Qué acción elige π ?
Exposición	¿Quién recibe la candidata y cuánto tráfico se mueve?
Observabilidad	¿Qué traza, métrica y log se guardan?
Gate	¿Qué condiciones permiten avanzar?
Recuperación	¿Cómo volvemos a la política de reserva?

OpenTelemetry define señales como trazas, métricas y logs para instrumentar sistemas distribuidos.⁶⁷⁸ En IA generativa, además, la semántica de GenAI ayuda a nombrar atributos de peticiones, modelos y operaciones de generación.⁹

El contrato matemático del serving

Una política servida se escribe con la notación estándar de RL, anotando además la versión, para separar decisión, versión y restricciones. No es una API ni una especificación de producto, sino una forma compacta de recordar qué debe viajar junto a una decisión operativa:

$$a_t \sim \pi_{\theta,v}(a \mid x_t, c_t)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
t	Instante o petición.	Petición 1523 de la ventana actual.
x_t	Contexto de decisión.	Pregunta, usuario agregado, documentos, tarea.
c_t	Restricciones operativas.	Slice permitido, coste máximo, permisos, flag.
a_t	Acción elegida.	<code>large_reasoning_model</code> .
$\pi_{\theta,v}$	Política con parámetros θ y versión v .	<code>policy_candidate_v4</code> .

La decisión no se guarda solo como acción. Se guarda como evento:

$$e_t = (x_t, a_t, p_t, r_t, g_t, v_\pi, v_R, s_t)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
p_t	Probabilidad o razón de selección.	0,05 en un piloto al 5 %.
r_t	Recompensa observada o estimada.	0,73.
g_t	Resultado de gates duros.	JSON válido, evidencia soportada.
v_π	Versión de política.	<code>policy_candidate_v4</code> .
v_R	Versión de reward card.	<code>1.0.0</code> .
s_t	Slice operativo.	<code>rag</code> , <code>sql</code> , <code>herramientas</code> .

Después agregamos por ventana:

$$SLI_m(W) = \frac{1}{|W|} \sum_{e_t \in W} h_m(e_t)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
W	Ventana de observación.	Últimas 12 horas.
m	Métrica o indicador.	<code>reward_mean</code> , <code>p95_latency_ms</code> .
$h_m(e_t)$	Función que extrae una medición del evento.	Latencia de la petición.
$SLI_m(W)$	Indicador medido en la ventana.	Reward medio 0,73.

Un SLO convierte ese indicador en objetivo:

$$SLI_m(W) \geq SLO_m$$

o, si menor es mejor:

$$SLI_m(W) \leq SLO_m$$

Google SRE insiste en separar monitorización útil, SLOs y alertas: no medimos por llenar gráficos, medimos para decidir y actuar.[101112](#)

En serving de políticas, el SLO no es solo disponibilidad. También hay SLOs de calidad: evidencia, reward, fallback, coste, latencia y slices.

Drift: cuando la ventana actual ya no se parece a la referencia

Drift significa que algo relevante cambió. Puede cambiar la distribución de entradas, la mezcla de slices, la acción elegida, la recompensa, el coste, el comportamiento de herramientas o el propio verificador.

Gama y otros revisan el problema de concept drift como cambio en flujos de datos donde la relación entre entrada y objetivo evoluciona con el tiempo.¹³ Sugiyama y otros trataron el covariate shift como el caso donde cambia la distribución de entradas aunque la relación condicional se mantenga.¹⁴

En este capítulo no queremos convertir al lector en especialista de detección de drift. Queremos que pueda mirar producción y hacerse estas preguntas:

TIPO DE DRIFT	QUÉ CAMBIA	EJEMPLO
Drift de contexto	Cambia $P(x)$.	Llegan más preguntas SQL que RAG.
Drift de acción	Cambia la mezcla de acciones.	La política llama mucho más al modelo caro.
Drift de recompensa	Cambia la relación entre reward y resultado real.	Sube reward, baja evidencia revisada.
Drift de coste	Cambia latencia, tokens o herramientas.	P95 pasa de 1900 ms a 3100 ms.
Drift de verificador	Cambia cómo puntúa un grader.	El verificador de citas acepta peor un nuevo formato.
Drift de slice	Un segmento pequeño se degrada.	<code>privacidad</code> cae mientras el global parece bien.

Evidently documenta drift comparando una ventana de referencia con una ventana actual y aplicando métodos distintos según tipo de columna, tamaño y cardinalidad.¹⁵ La herramienta concreta puede cambiar; la idea estable no: siempre necesitas referencia, actual, métrica y criterio de decisión.

PSI: una medida sencilla para empezar

Una métrica frecuente para comparar distribuciones discretizadas es el Population Stability Index:

$$PSI(P, Q) = \sum_{b=1}^B (q_b - p_b) \log \left(\frac{q_b}{p_b} \right)$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
P	Distribución de referencia.	Acciones de la política estable.
Q	Distribución actual.	Acciones de la candidata.
b	Bin o categoría.	rag , sql , herramientas .
p_b	Proporción de referencia en el bin.	0,55.
q_b	Proporción actual en el bin.	0,32.
B	Número de bins.	4 slices.

El PSI no es una verdad universal. Es una alarma. Si Q se separa mucho de P , toca mirar. Aquí usamos un umbral conservador de 0,20:

PSI	LECTURA PRÁCTICA
0,00-0,10	Cambio pequeño.
0,10-0,20	Revisar con cuidado.
> 0,20	No aumentar exposición sin explicación.

Ejemplo: si la distribución de acciones cambia de `small_rag_model` a `large_reasoning_model` , quizá el reward sube un poco, pero el coste y la latencia también. Una política candidata no puede avanzar solo porque una métrica mejore si está cambiando la forma de usar el sistema.

El gate de rollout

Un gate de serving combina SLOs, drift, trazas y preparación de rollback:

$$G_k = 1[\text{SLI}_{\text{reward}} \geq \tau_R] \cdot 1[\text{SLI}_{\text{evidencia}} \geq \tau_E] \cdot 1[\text{p95}_{\text{latencia}} \leq \tau_L] \cdot 1[\text{PSI} \leq \tau_D] \cdot 1[\text{rollback listo}]$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
G_k	Gate de la etapa k .	Gate para <code>pilot_5</code> .
τ_R	Umbral mínimo de reward.	0,62.
τ_E	Umbral mínimo de evidencia.	0,90.
τ_L	Máximo p95 de latencia.	2500 ms.
τ_D	Máximo drift aceptable.	PSI 0,20.

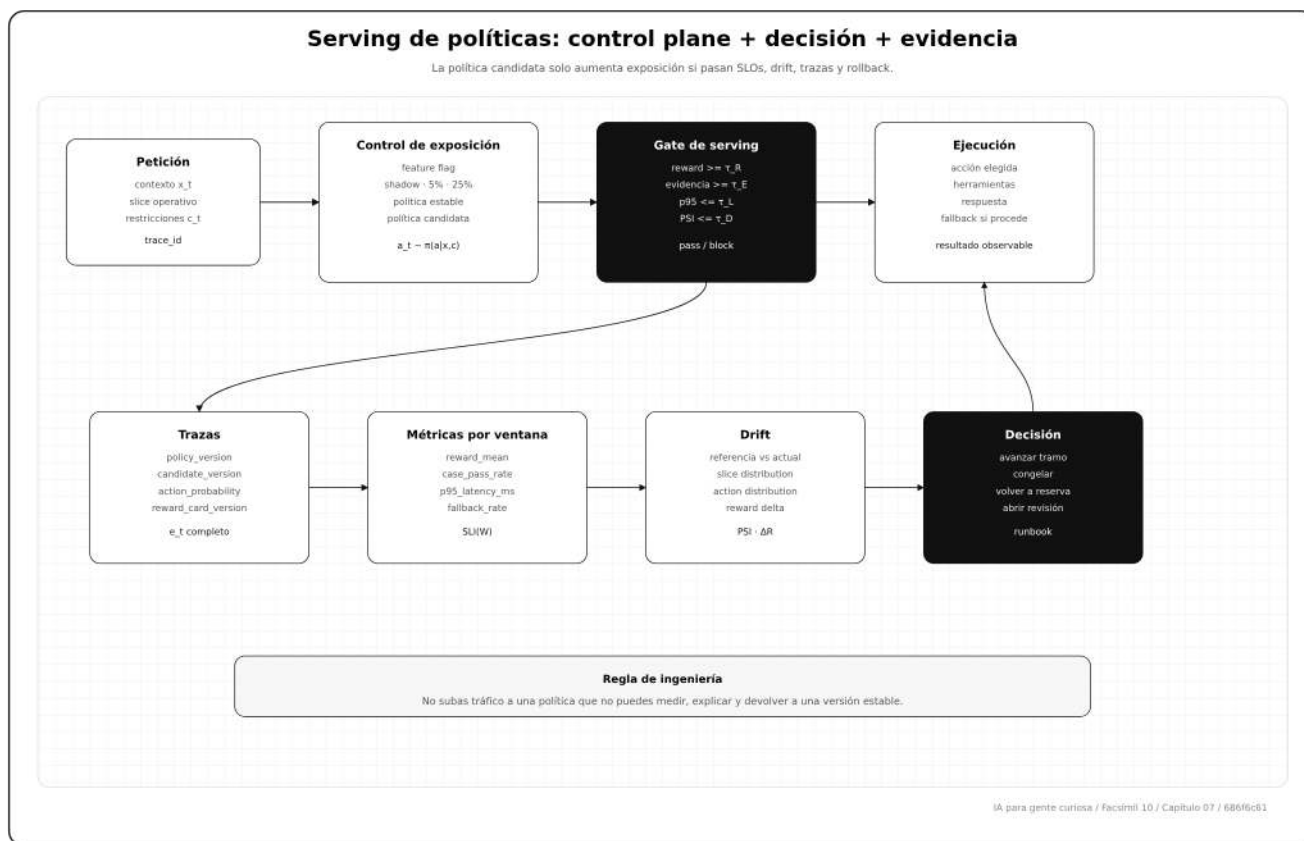
Si cualquier factor vale 0, el producto vale 0. Esa es la gracia: el rollout no avanza porque una media compense un fallo operativo. No compensamos rollback no preparado con reward alto. No compensamos evidencia baja con latencia buena. No compensamos drift fuerte con una media global bonita.

Una secuencia razonable:

ETAPA	TRÁFICO REAL	QUÉ EXIGE
Shadow	0 %	Trazas completas, paridad razonable, latencia medida.
Piloto 5 %	5 %	SLOs por slice y política de reserva lista.
Piloto 25 %	25 %	Drift bajo, coste dentro de presupuesto, revisión de soporte.
Piloto 50 %	50 %	Estabilidad en varias ventanas.
General	100 %	Runbook, alertas y deuda de flags revisada.

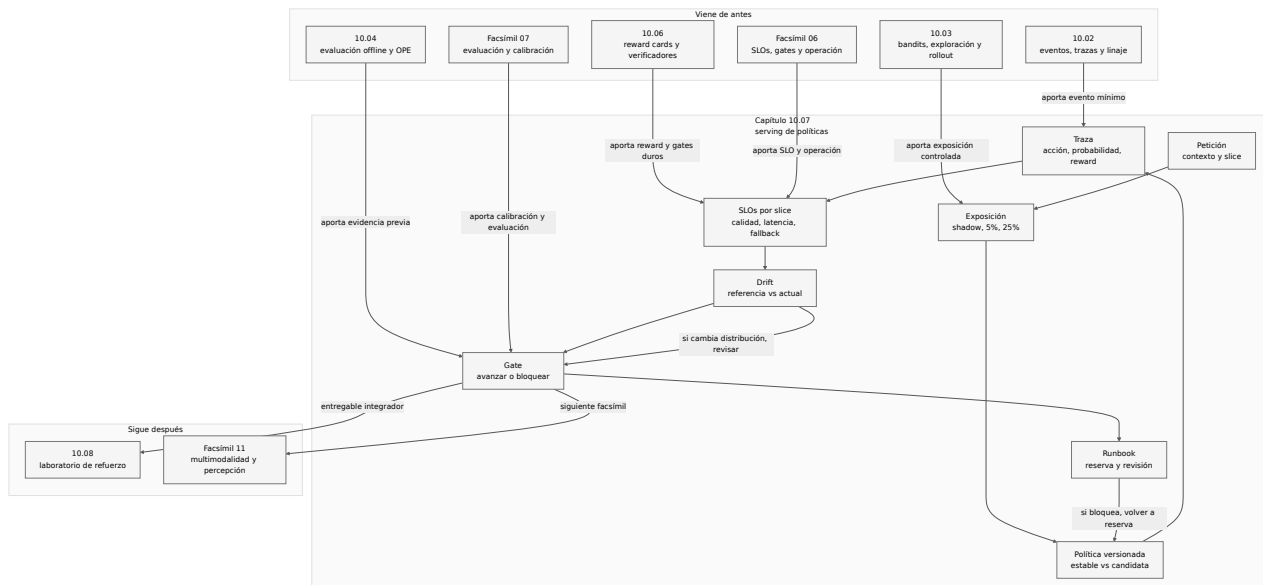
LaunchDarkly documenta rollouts porcentuales y progresivos como forma de liberar cambios gradualmente.¹⁶ También advierte sobre deuda técnica de flags: una flag que controla un rollout no debe quedarse indefinidamente como residuo operativo.¹⁷

La anatomía de un serving defendible



Cómo encaja todo

Lee este mapa como el cierre operativo del facsímil antes de la recapitulación final. Heredamos eventos y linaje del capítulo 02, exploración y rollout del 03, evaluación offline del 04 y reward cards del 06. Aquí juntamos todo en una decisión de serving: shadow, piloto, drift, SLOs, rollback y runbook.



Vocabulario aprendido

TÉRMINO	QUÉ SIGNIFICA	CÓMO LO USARÍA
Serving de políticas	Capa que ejecuta una política versionada con gates y trazas.	No confundir con un endpoint sin control operativo.
Política candidata	Versión que queremos probar.	<code>policy_candidate_v4</code> .
Política de reserva	Versión estable a la que podemos volver.	<code>policy_stable_v3</code> .
Shadow	Fase sin tráfico real de decisión.	La candidata recomienda, pero no decide.
Rollout	Aumento gradual de exposición.	5 %, 25 %, 50 %.
Drift	Cambio entre referencia y actual.	Cambio de slices, acciones o reward.
PSI	Métrica para comparar distribuciones.	Detectar cambio de mezcla de acciones.
SLI	Indicador medido.	<code>p95_latency_ms</code> .
SLO	Objetivo del indicador.	P95 menor o igual que 2500 ms.
Runbook	Guía operativa de actuación.	Qué mirar y qué hacer si bloquea.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Pensar que servir es desplegar un endpoint.	La infraestructura responde.	Exigir política versionada, trazas y gate.
Mirar solo una media global.	Es cómoda y fácil de enseñar.	Revisar slices y casos pequeños.
Aumentar tráfico sin shadow.	La evaluación offline salió bien.	Pasar primero por shadow con trazas completas.
No tener política de reserva.	Nadie piensa en volver hasta que hace falta.	Declararla antes de abrir piloto.
Confundir feature flag con política.	Ambas controlan comportamiento.	La flag controla exposición; la política decide acción.
Medir drift sin decisión asociada.	El gráfico parece suficiente.	Definir umbral, gate y runbook.

Antes de pasar página

- ¿Por qué una política no termina cuando pasa evaluación offline?
- ¿Qué diferencia hay entre política, endpoint y feature flag?
- ¿Qué campos guardarías en una traza de decisión?
- ¿Qué SLOs pondrías para un router de modelos?
- ¿Por qué el drift debe mirarse por slice?
- ¿Qué mide el PSI?
- ¿Qué debería ocurrir en shadow?
- ¿Qué condiciones bloquearían un piloto al 5 %?
- ¿Qué debe contener una política de reserva?
- ¿Qué hace el runbook si el gate bloquea?

Para saber más

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML Test Score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data*, 1123-1132. <https://doi.org/10.1109/BigData.2017.8258038>

Evidently AI. (2026). *Data drift*. https://docs.evidentlyai.com/metrics/explainer_drift

Ewaschuk, R. (2016). *Monitoring Distributed Systems*. En B. Beyer, C. Jones, J. Petoff y N. R. Murphy (eds.), *Site Reliability Engineering*. <https://sre.google/sre-book/monitoring-distributed-systems/>

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 1-37. <https://doi.org/10.1145/2523813>

Jones, C., Wilkes, J., Murphy, N., & Smith, C. (2016). *Service Level Objectives*. En B. Beyer, C. Jones, J. Petoff y N. R. Murphy (eds.), *Site Reliability Engineering*. <https://sre.google/sre-book/service-level-objectives/>

LaunchDarkly. (2026). *Releasing features with LaunchDarkly*. <https://launchdarkly.com/docs/home/releases/releasing>

OpenTelemetry. (2026). *Traces*. <https://opentelemetry.io/docs/concepts/signals/traces/>

OpenTelemetry. (2026). *Metrics*. <https://opentelemetry.io/docs/concepts/signals/metrics/>

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden Technical Debt in Machine Learning Systems. *Advances in Neural Information Processing Systems*, 28. https://papers.nips.cc/paper_files/paper/2015/hash/86df7dcfd896fcdf2674f757a2463eba-Abstract.html

Sugiyama, M., Krauledat, M., & Müller, K.-R. (2007). Covariate Shift Adaptation by Importance Weighted Cross Validation. *Journal of Machine Learning Research*, 8, 985-1005. <https://jmlr.csail.mit.edu/papers/v8/sugiyama07a.html>

En resumen

IDEA	QUÉ DEBES RECORDAR
Una política servida es un sistema operativo, no solo una función.	Necesita versión, trazas, SLOs, gates, exposición controlada y política de reserva.
El drift es una señal para decidir, no un gráfico decorativo.	Compara referencia y actual por slice, acciones y reward antes de aumentar tráfico.
El rollout debe ser reversible.	Shadow, pilotos y runbook importan tanto como la métrica de calidad.
El cierre del facsímil ya puede integrar todo.	Datos, reward card, política, serving y gates quedan listos para practicarse juntos.

Notas

1. OpenTelemetry. (2026). *Documentation*. <https://opentelemetry.io/docs/>. Consultado el 9 de junio de 2026. ↵
2. LaunchDarkly. (2026). *Releasing features with LaunchDarkly*. <https://launchdarkly.com/docs/home/releases/releasing>. Consultado el 9 de junio de 2026. ↵
3. Evidently AI. (2026). *Data drift*. https://docs.evidentlyai.com/metrics/explainer_drift. Consultado el 9 de junio de 2026. ↵
4. Sculley, D. y otros (2015). *Hidden Technical Debt in Machine Learning Systems*. *Advances in Neural Information Processing Systems*, 28. https://papers.nips.cc/paper_files/paper/2015/hash/86df7dcfd896fcdf2674f757a2463eba-Abstract.html ↵
5. Breck, E., Cai, S., Nielsen, E., Salib, M. y Sculley, D. (2017). The ML Test Score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data*, 1123-1132. <https://doi.org/10.1109/BigData.2017.8258038> ↵
6. OpenTelemetry. (2026). *Traces*. <https://opentelemetry.io/docs/concepts/signals/traces/>. Consultado el 9 de junio de 2026. ↵
7. OpenTelemetry. (2026). *Metrics*. <https://opentelemetry.io/docs/concepts/signals/metrics/>. Consultado el 9 de junio de 2026. ↵
8. OpenTelemetry. (2026). *Logs*. <https://opentelemetry.io/docs/concepts/signals/logs/>. Consultado el 9 de junio de 2026. ↵
9. OpenTelemetry. (2026). *Semantic conventions for generative AI systems*. <https://opentelemetry.io/docs/specs/semconv/gen-ai/>. Consultado el 9 de junio de 2026. ↵
10. Ewaschuk, R. (2016). *Monitoring Distributed Systems*. En B. Beyer, C. Jones, J. Petoff y N. R. Murphy (eds.), *Site Reliability Engineering*. <https://sre.google/sre-book/monitoring-distributed-systems/>. Consultado el 9 de junio de 2026. ↵
11. Jones, C., Wilkes, J., Murphy, N. y Smith, C. (2016). *Service Level Objectives*. En B. Beyer, C. Jones, J. Petoff y N. R. Murphy (eds.), *Site Reliability Engineering*. <https://sre.google/sre-book/service-level-objectives/>. Consultado el 9 de junio de 2026. ↵
12. Wilkinson, J. (2018). *Alerting on SLOs*. En B. Beyer, N. R. Murphy, D. Rensin, K. Kawahara y S. Thorne (eds.), *The Site Reliability Workbook*. <https://sre.google/workbook/alerting-on-slos/>. Consultado el 9 de junio de 2026. ↵
13. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. y Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 1-37. <https://doi.org/10.1145/2523813> ↵

- .4. Sugiyama, M., Krauledat, M. y Müller, K.-R. (2007). Covariate Shift Adaptation by Importance Weighted Cross Validation. *Journal of Machine Learning Research*, 8, 985-1005.
<https://jmlr.csail.mit.edu/papers/v8/sugiyama07a.html> ↵
- .5. Evidently AI, 2026. ↵
- .6. LaunchDarkly, 2026. ↵
- .7. LaunchDarkly. (2026). *Reducing technical debt from feature flags*.
<https://launchdarkly.com/docs/guides/flags/technical-debt>. Consultado el 9 de junio de 2026.
↵

Capítulo 08: Recapitulación de aprendizaje por refuerzo

Un facsímil sobre aprendizaje por refuerzo no debería terminar con una frase bonita sobre agentes que aprenden. Debería terminar con una carpeta que puedas ejecutar, una política que puedas comparar, una recompensa que puedas discutir y una decisión que puedas defender.

Ese es el objetivo de este cierre. No vamos a entrenar un modelo grande. Vamos a hacer algo más controlado y mucho más útil para aprender: construir un circuito pequeño donde se vean las piezas esenciales de RL aplicado a sistemas de IA.

La pregunta final del facsímil no es:

¿Qué algoritmo parece más sofisticado?

La pregunta final es:

¿Publicarías esta política, con esta recompensa, sobre estos datos, bajo estos límites?

Si no puedes contestar eso, todavía no tienes un sistema de refuerzo. Tienes una simulación suelta.

Qué deberías poder hacer al terminar

Este facsímil no pretende que salgas entrenando un modelo de razonamiento desde cero. Pretende algo más práctico: que puedas mirar cualquier sistema que optimiza comportamiento y preguntar qué política ejecuta, qué recompensa persigue, qué futuro ignora y qué evidencia deja.

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Formalizar una decisión secuencial.	Escribes S, A, P, R, γ , identificas estado, acción, transición, recompensa y horizonte.
Leer una política como contrato operativo.	Distingues la política diseñada, la política versionada, la política servida y la política de reserva.
Calcular retorno y valor.	Separas recompensa inmediata, retorno descontado, valor esperado y coste de oportunidad.
Validar exploración.	Mides regret, coste, cobertura de acciones y límites de tráfico.
Diseñar recompensas con cuidado.	Detectas términos que premian señales cómodas pero equivocadas.
Entender post-training moderno.	Separas SFT, DPO, RLHF, RLAIIF, RFT y RLVR por la señal que usan.
Evaluar sin desplegar a ciegas.	Usas trazas, propensión, replay, casos de prueba y gates.
Preparar una publicación limitada.	Defines shadow, piloto, rollback, drift, SLI, SLO y runbook.

La idea de cierre:

RL es ingeniería de consecuencias. No basta con que una acción parezca buena: hay que medir qué futuro abre y qué coste deja detrás.

Lo que hemos construido

CAPÍTULO	PREGUNTA	ARTEFACTO TÉCNICO
01	¿Qué significa decidir por interacción?	MDP, política, retorno, valor y Bellman.
02	¿Qué datos hacen posible aprender de interacción?	Eventos, trayectorias, linaje, propensión y replay buffer.
03	¿Cómo aprende una política sin dejar de actuar?	Bandits, exploración, regret, UCB y límites de exposición.
04	¿Cómo evaluaríamos sin desplegar?	Evaluación offline, contrafactual, OPE y sesgo de logging.
05	¿Cómo se conectan preferencias y post-training?	SFT, pares de preferencia, recompensa, DPO, RLHF, RLVR.
06	¿Cómo diseñamos recompensas defendibles?	Reward cards, verificadores, casos negativos y gates.
07	¿Cómo vive una política en producción?	Serving, monitorización, drift, trazas y rollout.
08	¿Cómo lo practico sin entrenar un LLM?	Simulador, auditoría, entrega reproducible y decisión de publicación.

El patrón completo es este:

1. Definir el problema como decisión.
2. Registrar datos de interacción.
3. Comparar políticas.
4. Diseñar o auditar la recompensa.
5. Validar antes de publicar.
6. Publicar con límites.
7. Medir si el mundo cambió.

Fórmulas mínimas que debes llevarte

Un cierre de RL sin fórmulas se queda blando. No hace falta convertir cada práctica en un paper, pero sí conviene tener una chuleta mental. Estas son las piezas que aparecen una y otra vez.

MDP

$$\mathcal{M} = (S, A, P, R, \gamma)$$

SÍMBOLO	LECTURA
S	Conjunto de estados observables o parcialmente observables.
A	Conjunto de acciones posibles.
$P(s' \mid s, a)$	Dinámica: probabilidad de pasar al siguiente estado.
$R(s, a, s')$	Recompensa asociada a la transición.
γ	Factor de descuento entre presente y futuro.

La pregunta de ingeniería es: ¿de verdad tengo estado, acción, consecuencia y horizonte? Si solo tengo una clasificación aislada, quizá no necesito RL. Si tengo decisiones repetidas con consecuencias acumuladas, entonces sí empieza a oler a problema secuencial.

Política

$$a_t \sim \pi_{\theta,v}(a \mid s_t, c_t)$$

La política no es una idea abstracta. En producción debería tener versión, contexto, parámetros, trazas y política de reserva. Cuando alguien dice “el sistema decidió”, un ingeniero debería poder preguntar: ¿qué versión de la política?, ¿con qué entrada?, ¿con qué probabilidad?, ¿contra qué alternativa?, ¿con qué gate?

Retorno

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

La recompensa inmediata puede ser buena y el retorno malo. Un sistema puede ahorrar tokens hoy y crear más trabajo mañana. Puede resolver una pregunta de forma rápida y aumentar reclamaciones. Por eso el retorno fuerza a pensar en consecuencias, no solo en el marcador del instante.

Regret

$$\text{Regret}_T = T\mu^* - \sum_{t=1}^T r_t$$

SÍMBOLO	LECTURA
T	Número de rondas.
μ^*	Recompensa media de la mejor acción conocida en el escenario.
r_t	Recompensa obtenida en la ronda t .

El regret traduce una intuición muy práctica: cuánto te ha costado aprender. Una política puede estar aprendiendo, pero si el coste de exploración es demasiado alto para el dominio, no debería exponerse igual que en una demo.

Recompensa compuesta

$$R(x, y) = w_c C(x, y) + w_e E(x, y) + w_a A(x, y) + w_f F(x, y) - w_k K(x, y)$$

TÉRMINO	LECTURA EN LA PRÁCTICA
C	Corrección de la respuesta.
E	Evidencia o cita válida.
A	Abstención cuando falta fuente suficiente.
F	Formato validable.
K	Coste de tokens y herramientas.

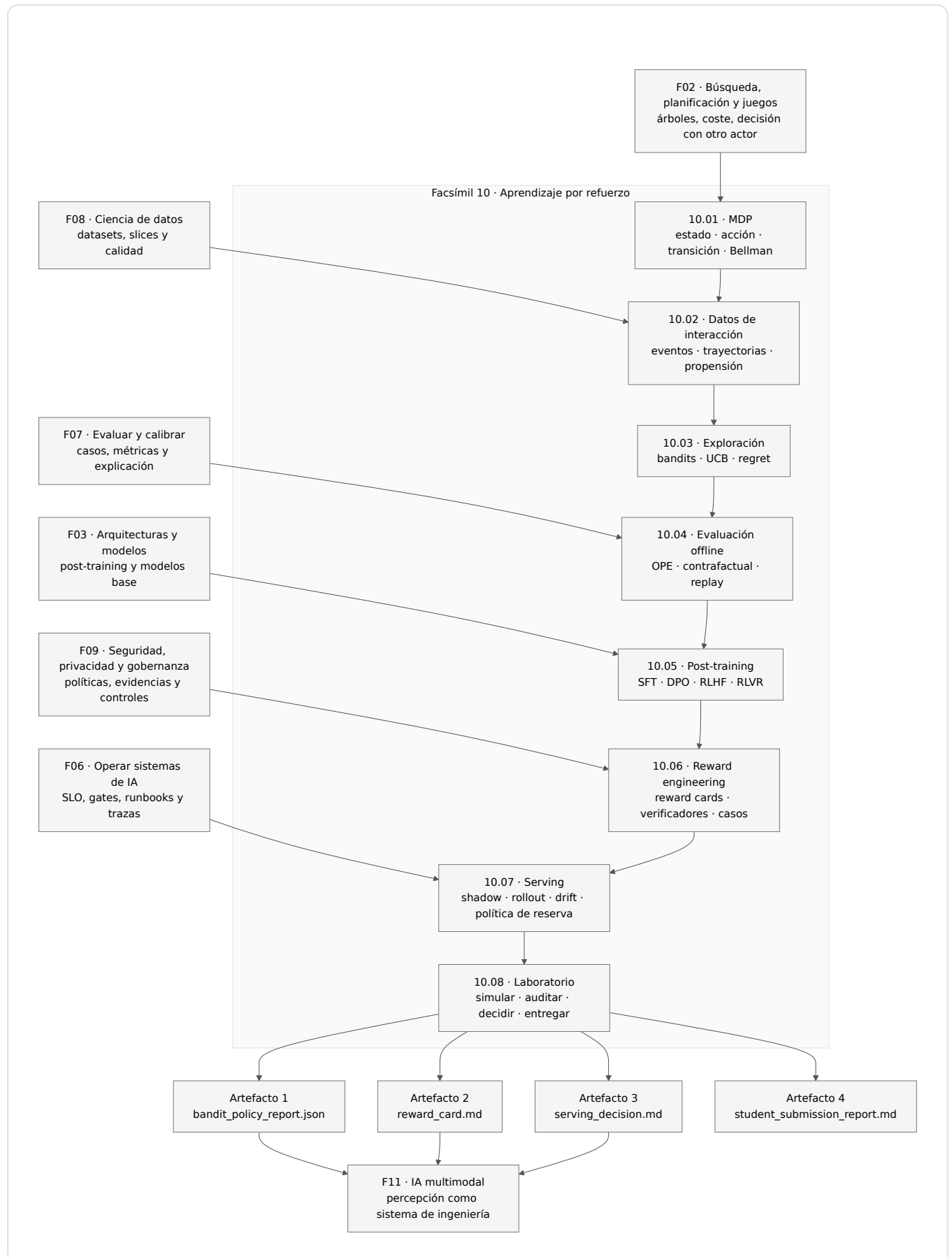
La recompensa compuesta obliga a poner números donde antes había intención. Eso no la vuelve perfecta. La vuelve revisable.

Gate de publicación

$$\text{gate}(\pi_v) = \mathbf{1}[R_{acum} \geq \tau_R \wedge \text{Regret} \leq \tau_{\text{regret}} \wedge \text{trazas} \geq \tau_T \wedge \text{drift} \leq \tau_D]$$

Un gate no sustituye criterio. Lo documenta. Si una política no deja trazas, no mide regret, no tiene política de reserva o no sabe detectar drift, no está lista aunque el promedio parezca atractivo.

Cómo encaja todo



Cuadernos para practicar

Has operado políticas a lo largo del facsímil; estos cuadernos te dejan ver nacer la decisión desde las recompensas. Son *notebooks* que se abren en Google Colab —gratis, en el navegador— o te puedes descargar. Cada uno, explicado paso a paso, con salidas reales, y anclado al capítulo del que sale.

MDP, Bellman y un mundo de casillas

Qué prácticas: resolver un mundo de casillas con *value iteration* y ver emerger la política óptima. **Dónde encaja:** capítulo 1 (MDP, políticas, retorno y Bellman). **Qué necesitas:** un navegador. Corre en CPU; sin claves.

Sueltas un robot en un mundo de 3×4 con una salida (+1), una trampa (-1) y un muro. No le enseñas el camino: solo le das las recompensas (y un coste de -0,04 por paso) y dejas que la ecuación de Bellman calcule, iterando, lo que vale estar en cada casilla. En **15 iteraciones** los valores convergen (van de +0,51 cerca del inicio a +1,00 en la salida) y la política sale sola: las flechas rodean el muro y se alejan de la trampa, porque su valor negativo se propaga a los vecinos. Nadie programó la ruta; emergió del valor.

Decidir bajo incertidumbre con consecuencias que se encadenan es lo que hacen un robot, un recomendador o un agente que planea. *Value iteration* es la forma más limpia de ver cómo «el valor de aquí» depende del «valor de lo que viene».

► [Abrir en Google Colab](#)

↓ [Descargar el cuaderno \(.ipynb\)](#)

Explorar o explotar: el dilema de los bandidos

Qué prácticas: medir, en *regret*, tres estrategias para aprender cuál de varias opciones es la mejor. **Dónde encaja:** capítulo 3 (exploración, *bandits* y validación de políticas). **Qué necesitas:** un navegador. Corre en CPU; sin claves.

Tienes seis máquinas con probabilidades de premio desconocidas y 2000 tiradas. ¿Sigues con la que parece buena (explotar) o pruebas otras por si lo son más (explorar)? Comparas cuatro formas de decidir por su *regret* —lo que pierdes frente a haber jugado siempre la mejor—. El azar acumula **608** de *regret* (no aprende nada); ϵ -greedy, UCB y Thompson lo dejan muy atrás (**63, 114 y 45**) porque descubren la buena y se quedan con ella. La diferencia entre ganar y malgastar está en explorar lo justo.

Este dilema vive en cualquier sistema que aprende mientras actúa: tests A/B, recomendaciones, anuncios, dosis. Explotar de más te ancla en lo mediocre; explorar de más malgasta.

► **Abrir en Google Colab**

↓ **Descargar el cuaderno (.ipynb)**

Dónde solía tropezar yo

TROIEZO	POR QUÉ OCURRE	ANTÍDOTO
Pensar que RL exige entrenar un modelo grande	La literatura impresiona.	Empezar con simulaciones pequeñas y decisiones reproducibles.
Confundir recompensa con métrica de dashboard	Ambas son números.	Escribir qué comportamiento induce la recompensa.
No medir regret	Solo miramos recompensa final.	Comparar contra la mejor acción conocida o una referencia estable.
Usar verificadores incompletos	Pasan rápido y dan sensación de seguridad.	Crear casos negativos, slices y cobertura mínima.
Olvidar la política de reserva	La política cambia con datos.	Mantener versión estable y condición de retorno.
Publicar por promedio	El promedio tapa slices dañados o drift.	Medir por ventana, slice y acción.

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Laboratorio de refuerzo	Práctica reproducible para simular una política, auditar una recompensa y defender una decisión.
Bandit	Problema donde se elige repetidamente entre acciones y se observa recompensa de la acción elegida.
Regret	Coste acumulado de no haber elegido siempre la mejor acción disponible bajo el escenario de referencia.
Reward card	Documento técnico que explica objetivo, términos, pesos, casos, límites y gates de una recompensa.
Gate de política	Regla ejecutable que decide si una política puede avanzar, bloquearse o quedarse en revisión.
Modo sombra	Ejecución sin afectar la decisión real para comparar comportamiento y trazas antes del piloto.
Política de reserva	Versión estable a la que se vuelve si la candidata degrada calidad, coste, trazas o seguridad.
Entrega reproducible	Carpeta con comandos, datos, salidas y decisión que otra persona puede ejecutar y revisar.

Antes de pasar página

- ¿Qué significa que RL sea ingeniería de consecuencias?
- ¿Por qué un bandit puede ser más adecuado que un MDP completo?
- ¿Qué diferencia hay entre recompensa acumulada, retorno y regret?
- ¿Qué debe contener una reward card?
- ¿Por qué RLVR depende tanto de la calidad del verificador?
- ¿Qué evidencia pedirías antes de publicar una política candidata?
- ¿Qué conexión hay entre este facsímil y producto/UX?

Recursos para seguir: leer, construir y experimentar

El aprendizaje por refuerzo cambia la pregunta: ya no es «¿la salida es correcta?», sino «¿qué pasa después de actuar?». Has visto MDP, políticas, bandits, RLHF y reward engineering. Aquí tienes por dónde seguir.

Para experimentar sin código. En la caja «Pruébalo en 5 minutos» del primer capítulo lo viste: la demo GridWorld de Andrej Karpathy (cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html) deja ver la ecuación de Bellman trabajando, cómo los valores se extienden desde las recompensas y cómo emerge la política óptima a base de iterar. Es una de las formas más claras de entender qué significa «valor» en refuerzo.

Para construir. El entorno estándar para experimentar con RL es Gymnasium (de la Farama Foundation, sucesor de OpenAI Gym), que ofrece entornos listos para entrenar agentes; y librerías como Stable-Baselines3 o CleanRL para los algoritmos. Para el post-training de modelos de lenguaje (RLHF, DPO), las librerías de la comunidad de Hugging Face son el punto de partida.

Para leer. La referencia canónica e insustituible es *Reinforcement Learning: An Introduction*, de Sutton y Barto, disponible gratis en la web de los autores. Cada capítulo enlaza sus fuentes en «Para saber más». Conectar la teoría con la operación de una política es lo que separa la ecuación de Bellman de un sistema que decide con consecuencias.

Para saber más

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press. <https://incompleteideas.net/book/the-book-2nd.html>

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. arXiv:1707.06347. <https://arxiv.org/abs/1707.06347>

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Aspell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). *Training Language Models to Follow Instructions with Human Feedback*. arXiv:2203.02155. <https://arxiv.org/abs/2203.02155>

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., & Finn, C. (2023). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv:2305.18290. <https://arxiv.org/abs/2305.18290>

Bai, Y., Kadavath, S., Kundu, S., Aspell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., ... Kaplan, J. (2022). *Constitutional AI: Harmlessness from AI Feedback*. arXiv:2212.08073. <https://arxiv.org/abs/2212.08073>

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., ... Liang, W. (2025). *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. arXiv:2501.12948. <https://arxiv.org/abs/2501.12948>

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML Test Score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data*, 1123-1132. <https://doi.org/10.1109/BigData.2017.8258038>

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 1-37. <https://doi.org/10.1145/2523813>

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden Technical Debt in Machine Learning Systems. *Advances in Neural Information Processing Systems*, 28. https://papers.nips.cc/paper_files/paper/2015/hash/86df7dcfd896fc2674f757a2463eba-Abstract.html

1

2

3

En resumen

IDEA	QUÉ DEBES RECORDAR
MDP	Si no puedes definir estado, acción, transición, recompensa y descuento, quizá no tienes un problema RL formal.
Política	Es la regla que realmente se ejecuta, no la que aparece en una presentación.
Retorno	El futuro cuenta; ignorarlo crea decisiones miopes.
Bandits	Son útiles cuando eliges repetidamente entre opciones y puedes medir recompensa.
Regret	Aprender tiene coste; medirlo evita decisiones ingenuas.
Reward design	La recompensa es una especificación de producto e ingeniería.
Post-training	El método depende de la señal: ejemplos, preferencias, recompensa o verificador.
Serving	Una política no se publica: se expone gradualmente, se observa y se puede detener.
Entrega	La entrega buena no solo obtiene resultados; deja evidencia para que otra persona los revise.

Notas

1. Sutton y Barto (2018) es la referencia de base para separar MDP, política, valor, retorno y exploración sin mezclarlo con marketing de agentes. ↵
2. Ouyang y otros (2022), Rafailov y otros (2023) y Bai y otros (2022) ayudan a conectar el cierre con post-training moderno: feedback humano, preferencias directas y feedback asistido por IA. ↵
3. Breck y otros (2017), Gama y otros (2014) y Sculley y otros (2015) sostienen la parte de operación: readiness, drift y deuda técnica en sistemas ML. ↵