

Intro a RLHF

Lucca Frachelle

septiembre 2026

Referencia principal: [Ouyang et al. \(2022\)](#). Lectura complementaria: [RLHF Book](#) (Nathan Lambert).

De language model a policy de RL

Un *large language model* no almacena respuestas prefabricadas. Parametriza una distribución de probabilidad sobre secuencias de tokens: dado un prompt x , asigna probabilidades a posibles respuestas y mediante $\pi_\theta(y \mid x)$, con θ en el orden de miles de millones de parámetros distribuidos en un Transformer. Durante el **pre-entrenamiento**, esos pesos se ajustan para maximizar la verosimilitud de texto observado en corpus masivos. El resultado es un modelo con amplio conocimiento lingüístico y factual en sentido estadístico, pero cuyo objetivo original — predecir el siguiente token del internet — no coincide con el de un asistente que siga instrucciones de forma útil y segura.

Las respuestas se generan de forma **autoregresiva**. En post-entrenamiento conviene escribir prompt x y respuesta $y = (y_1, \dots, y_T)$, con

$$\pi_\theta(y \mid x) = \prod_{t=1}^T \pi_\theta(y_t \mid x, y_{<t}).$$

En **pre-entrenamiento**, en cambio, no hay split prompt/respuesta: sobre una secuencia cruda del corpus $w = (w_1, \dots, w_N)$ se minimiza

$$\mathcal{L}_{\text{LM}}(\theta) = - \sum_{t=1}^N \log \pi_\theta(w_t \mid w_{<t}),$$

equivalente a maximizar la verosimilitud de cada token condicionado a todo el contexto previo — que puede cruzar frases y documentos. En cada paso, el Transformer produce logits sobre el vocabulario y aplica un softmax; la arquitectura concreta (atención, capas, escala) importa menos, para nuestros propósitos, que el hecho de que π_θ es una policy paramétrica diferenciable cuyas acciones son tokens discretos.

Desde la perspectiva del TAR, conviene leer el LLM como una policy sobre un espacio de acciones enorme. El estado en el paso t es el contexto $(x, y_{<t})$; la acción es el token y_t ; la trayectoria es la respuesta completa y ; un episodio consiste en un prompt y la secuencia generada hasta el fin. A diferencia de un MDP clásico, no hay una dinámica de entorno $P(s' \mid s, a)$ independiente de la policy: el contexto solo crece porque el propio modelo va appendeando tokens. Tampoco hay, en la formulación de InstructGPT, un reward denso en cada paso intermedio: el Reward Model evalúa la respuesta terminada. La situación es la de un **bandit contextual**: se observa un contexto x , se elige una acción de alta dimensión y , y se recibe un escalar $r_\phi(x, y)$.

El **post-entrenamiento** no reemplaza el pre-entrenamiento sino que parte de él. Donde el pre-training optimizaba $\log \pi_\theta(y \mid x)$ sobre texto crudo, InstructGPT optimiza una combinación del score de un Reward Model y una penalización que mide cuánto se aleja la policy del checkpoint de supervised fine-tuning. El pre-entrenamiento aporta la capacidad lingüística base; el post-entrenamiento reorienta el

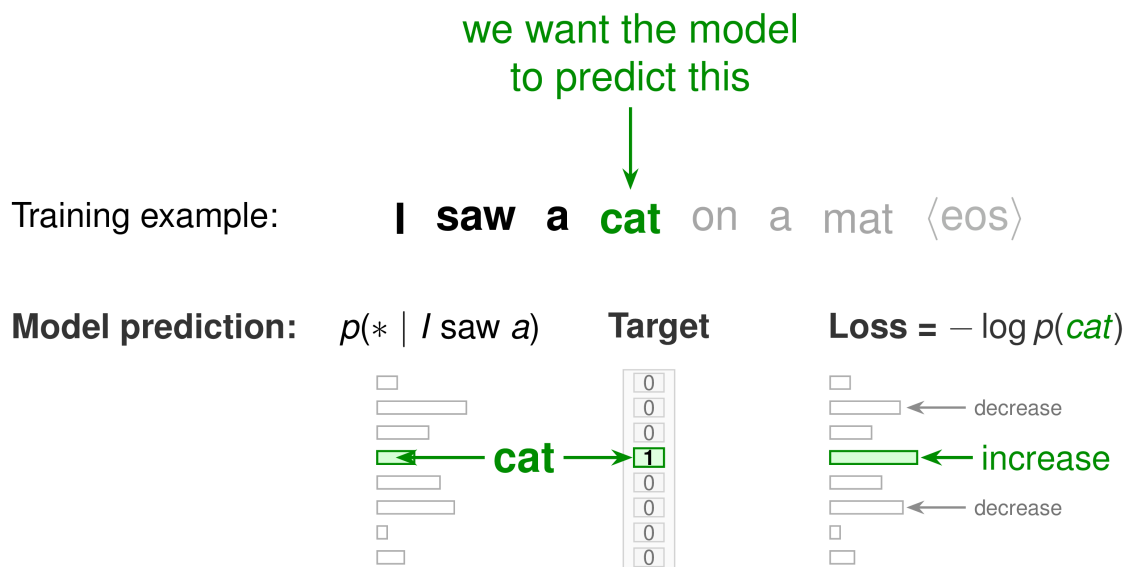


Figura 1: Predicción del siguiente token en pre-entrenamiento.

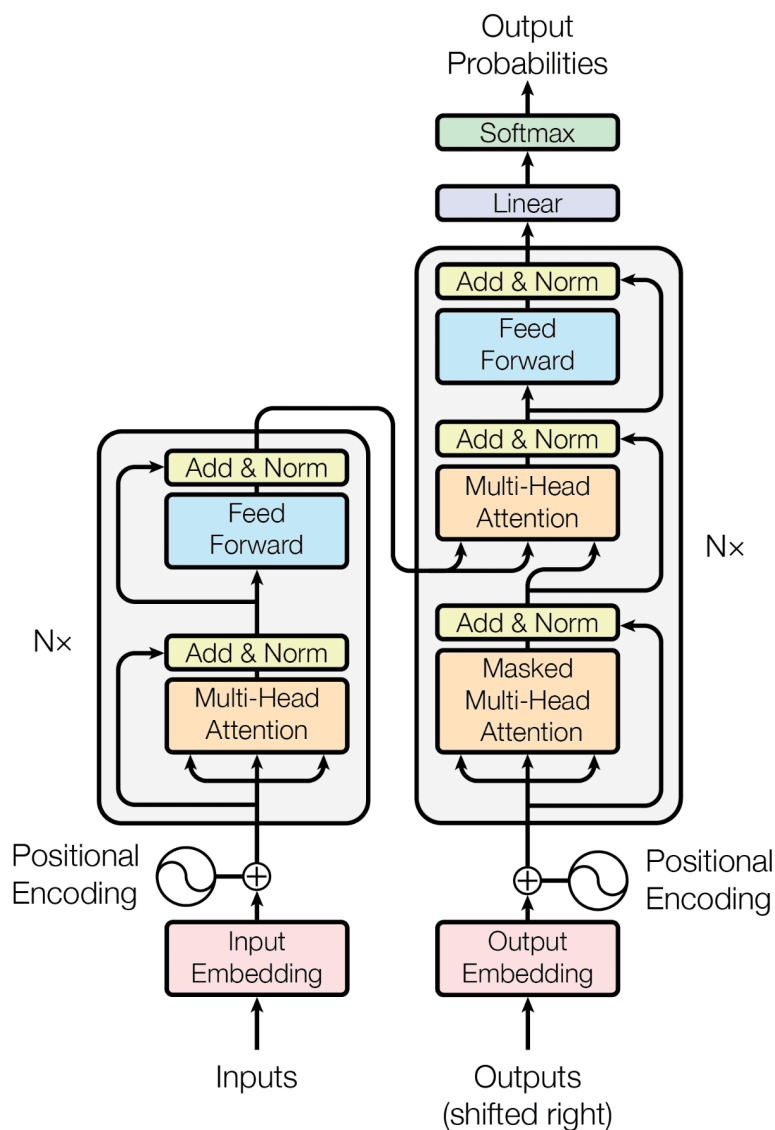


Figura 2: Arquitectura Transformer.

comportamiento hacia el de un asistente.

El pipeline de InstructGPT y sus datos

Ouyang et al. entrenan variantes de GPT-3 (1.3B, 6B y 175B parámetros) en tres etapas encadenadas: supervised fine-tuning (SFT), entrenamiento de un Reward Model (RM) sobre preferencias humanas, y fine-tuning por RL con PPO sobre ese RM. Salvo indicación contraria, cuando el paper habla de *InstructGPT* se refiere a la variante **PPO-ptx**, que mezcla gradientes de pretraining con los de PPO; volveremos sobre eso en la sección correspondiente.

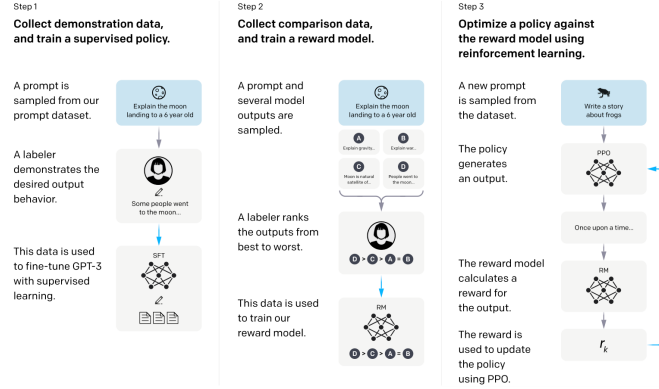


Figura 3: Pipeline InstructGPT (Fig. 2 del paper).

Los datos provienen de dos fuentes. Por un lado, unos cuarenta anotadores contratados escribieron instrucciones de distintos tipos — prompts simples, prompts con ejemplos few-shot, y casos de uso inspirados en la waitlist de la API de OpenAI. Por otro, se recolectaron prompts reales enviados por clientes a versiones tempranas de la API. La mayoría de las tareas son generativas (preguntas abiertas, diálogo, resumen, etc.); más del 96 % del material está en inglés.

A partir de esos prompts se construyen tres conjuntos de entrenamiento, cada uno con un rol distinto en el pipeline. El dataset **SFT** contiene unas 13 mil demostraciones (x, y^*) : el anotador escribe la respuesta que considera ideal. El dataset **RM** reúne unos 33 mil prompts para los cuales se generaron entre cuatro y nueve respuestas del modelo y los humanos produjeron un ranking completo. El dataset **PPO** aporta unos 31 mil prompts de la API sin etiquetas adicionales: solo sirven como contextos para rollouts durante el RL. Ninguno de estos conjuntos es público.

La recolección de preferencias merece detenerse un instante. Para cada prompt del dataset RM, el modelo genera K completions distintas (con $K \in [4, 9]$) y el anotador las ordena de mejor a peor, obteniendo $y_1 \succ y_2 \succ \dots \succ y_K$. Ese ranking único induce hasta $\binom{K}{2}$ comparaciones pareadas consistentes — si y_i está por encima de y_j en el ranking, se registra $y_i \succ y_j$. En la práctica hay muchos más pares de comparación que prompts, lo cual enriquece el entrenamiento del RM aun con un volumen moderado de anotación humana.

Supervised fine-tuning

La primera etapa es la más directa. Sobre las demostraciones humanas $\{(x, y^*)\}$ se minimiza la negative log-likelihood de la respuesta demostrada, token a token:

$$\mathcal{L}_{\text{SFT}}(\theta) = - \sum_{(x, y^*)} \sum_{t=1}^{|y^*|} \log \pi_{\theta}(y_t^* \mid x, y_{<t}^*).$$

Formalmente, es el mismo objetivo del language modeling aplicado a pares instrucción-respuesta. El gradiente incrementa la probabilidad de la trayectoria y^* bajo π_{θ} . No hay señal relativa entre dos

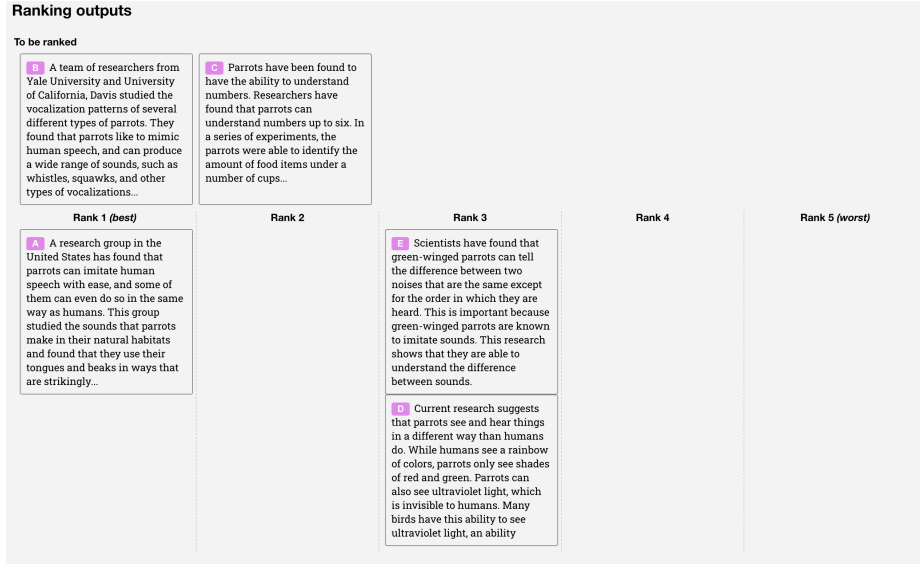


Figura 4: Interfaz de ranking (Fig. 12).

respuestas alternativas: el modelo aprende a imitar un ejemplo dado, no a preferir una respuesta sobre otra cuando ambas serían plausibles.

Eso tiene consecuencias claras. El SFT enseña al base model a adoptar el *formato* de asistente — responder a instrucciones en lugar de continuar texto arbitrariamente — y condiciona la distribución hacia un subespacio de respuestas razonables. Pero no resuelve el problema de la alineación fina: dos completions distintas pueden ser gramaticalmente correctas y una ser claramente superior para el usuario. Esa distinción requiere preferencias y, más adelante, RL.

En el paper hay dos corridas de SFT con propósitos distintos. Para las evaluaciones de baseline, entrenan durante 16 épocas con dropout 0.2 y learning rate cosine; curiosamente, la validation loss empeora tras la primera época, pero el checkpoint se selecciona por el score del RM en validación, no por la loss de SFT. Para inicializar el RL, en cambio, usan una corrida más corta de dos épocas con un 10 % de datos de pretraining mezclados. Ese checkpoint es π^{SFT} : sirve como punto de partida de la policy π_{ϕ}^{RL} y, crucialmente, como ancla en la penalización KL de la etapa de PPO.

El Reward Model y el modelo Bradley-Terry

La segunda etapa traduce juicios humanos en una función de reward diferenciable $r_{\phi}(x, y) \in \mathbb{R}$. El paper sigue la receta de Stiennon et al. (2020), aplicada antes al resumen de textos, y la extiende a instrucciones generales. La pieza conceptual clave es el **modelo Bradley-Terry** (Bradley y Terry, 1952).

La idea parte de postular que, fijado un prompt x , cada respuesta y tiene una utilidad latente $u(x, y)$ que resume cuán deseable es para un anotador representativo. Esa utilidad no se observa; lo que se observa son comparaciones. Bradley y Terry modelan la probabilidad de que un anotador prefiera y_a sobre y_b como una función que depende únicamente de la diferencia de utilidades:

$$P(y_a \succ y_b \mid x) = \frac{e^{u(x, y_a)}}{e^{u(x, y_a)} + e^{u(x, y_b)}} = \sigma(u(x, y_a) - u(x, y_b)),$$

donde $\sigma(z) = 1/(1+e^{-z})$ es la función logística. Si $u(x, y_a)$ supera ampliamente a $u(x, y_b)$, la preferencia por y_a es casi segura; si las utilidades son similares, la elección se aproxima al azar. La forma logística no es arbitraria: emerge en modelos de elección discreta cuando las utilidades están perturbadas por ruido de tipo extreme value (la derivación clásica del logit).

InstructGPT parametriza $u(x, y) \approx r_{\phi}(x, y)$ con una red neuronal. Concretamente, toman un GPT-3

de 6B parámetros — un solo RM para todas las políticas, porque la versión de 175B resultó inestable como función de valor durante el RL —, eliminan la capa de unembedding y la reemplazan por una proyección escalar. El modelo recibe el prompt y la respuesta concatenados y devuelve un número. Para una comparación observada (y_w, y_l) , con y_w el *winner* y y_l el *loser*, la probabilidad modelada de la preferencia registrada es

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

Una propiedad importante del modelo es la **invarianza a traslación**: si se suma una constante a todos los scores, las probabilidades no cambian, porque solo importan las diferencias. Por eso, antes de pasar al RL, el paper normaliza el RM de modo que las demostraciones SFT tengan score medio cero. Eso fija la escala que de otro modo sería indeterminada.

Los rankings de K respuestas se convierten en datos de entrenamiento agrupando, para cada prompt, todas las comparaciones $y_i \succ y_j$ con $i < j$. El paper advierte que mezclar al azar pares provenientes del mismo prompt en distintos batches provoca overfit severo en una sola época, porque las comparaciones dentro de un ranking están fuertemente correlacionadas. Por ello entrenan con un elemento de batch por prompt que contiene el conjunto completo de comparaciones de ese prompt — hasta $K^2 \leq 2304$ pares cuando el batch incluye 64 prompts.

La estimación es por máxima verosimilitud. Sea $\mathcal{D}$ la distribución sobre tuplas (x, y_w, y_l) inducida por los rankings. La log-verosimilitud del modelo Bradley-Terry es

$$\ell(\phi) = \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) \right],$$

y el entrenamiento minimiza su opuesto, que el paper escribe como la ecuación (1):

$$\mathcal{L}_{\text{RM}}(\phi) = -\frac{1}{K^2} \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) \right]. \quad (1)$$

El factor $1/K^2$ promedia explícitamente sobre las comparaciones asociadas a cada tarea de anotación. Esta pérdida no es un clasificador binario genérico pegado sobre scores: es la verosimilitud de un modelo probabilístico de preferencias, y maximizarla ajusta r_ϕ para que el orden inducido por los scores coincida, en promedio, con el orden humano.

La intuición del gradiente ayuda a leer la ecuación. Para un par (y_w, y_l) , sea $d = r_\phi(x, y_w) - r_\phi(x, y_l)$. La derivada de $-\log \sigma(d)$ respecto de d es $1 - \sigma(d)$. Si el RM clasifica mal el par ($d \ll 0$), el gradiente es grande y empuja a separar los scores; si el par ya está bien clasificado ($d \gg 0$), el gradiente es casi nulo y no hace falta seguir amplificando la diferencia. El RM se entrena una sola época sobre unos 33 mil prompts; más épocas degradan la validación.

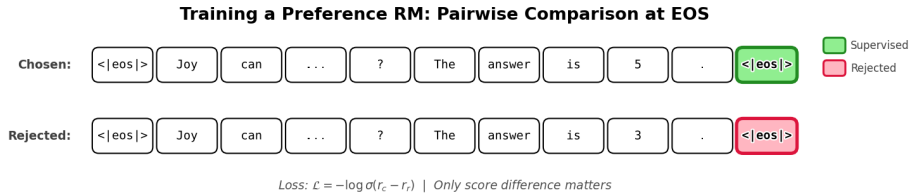


Figura 5: Entrenamiento del reward model.

Hay que insistir en que r_ϕ es un **proxy**. Aprende a reproducir preferencias humanas en la distribución de entrenamiento, pero puede ser explotado por una policy de RL que optimice demasiado agresivamente su score. Respuestas verbosas, aduladoras o con patrones que el RM premia sin aportar utilidad real son un riesgo conocido (*reward hacking*). Eso motiva la tercera etapa: no basta con maximizar r_ϕ ; hace falta regularizar.

PPO, la penalización KL y PPO-ptx

Con r_ϕ y π^{SFT} congelados, la tercera etapa fine-tunea una policy π_ϕ^{RL} mediante PPO. Cada episodio del bandit consiste en muestrear un prompt x del dataset PPO, generar una respuesta $y \sim \pi_\phi^{\text{RL}}(\cdot | x)$, evaluar el reward y terminar. El critic V se inicializa desde el RM (misma arquitectura de 6B) y las ventajas se estiman con GAE, con clipping $\varepsilon = 0.2$ como en el PPO estándar del curso.

La variante **PPO** sin mezcla de pretraining ($\gamma = 0$) maximiza

$$\text{objective}(\phi) = \mathbb{E}_{(x,y) \sim D_{\pi_\phi^{\text{RL}}}} \left[r_\phi(x, y) - \beta \log \frac{\pi_\phi^{\text{RL}}(y | x)}{\pi^{\text{SFT}}(y | x)} \right]. \quad (2a)$$

El primer término empuja hacia respuestas con alto score del RM, es decir, hacia lo que los humanos habrían preferido en promedio. El segundo penaliza desviarse de π^{SFT} . Como la probabilidad de una secuencia factoriza token a token, el log-ratio de la ecuación se descompone en una suma de contribuciones por paso; en la implementación del paper, esa penalización se aplica de forma **token a token** durante el rollout, lo que proporciona señal densa al critic aun cuando r_ϕ solo evalúa la respuesta completa. El coeficiente reportado es $\beta = 0.02$.

Maximizar únicamente $r_\phi(x, y)$ sería peligroso. La policy podría colapsar hacia completions que maximizan el proxy humano pero tienen probabilidad casi nula bajo el modelo pre-entrenado — lenguaje incoherente, fuera de distribución o explícitamente diseñado para engañar al RM. La penalización KL mantiene π_ϕ^{RL} cerca del manifold de lenguaje fluido aprendido en SFT. No es un detalle menor: sin ella, el RL sobre un RM imperfecto tiende a producir *reward hacking*.

Sin embargo, la KL por sí sola no resuelve otro problema. PPO puro mejora las preferencias humanas en la distribución de prompts de la API, pero **empeora** en benchmarks NLP estándar — SQuADv2, DROP, HellaSwag, traducción, entre otros. Ouyang et al. llaman a esta degradación *alignment tax*: al optimizar el proxy de alineación, la policy pierde parte de la capacidad lingüística general heredada del pretraining. Subir β atenúa la regresión, pero también limita cuánto puede mejorar el score del RM; no hay un knob que resuelva ambos extremos a la vez.

La solución que adopta InstructGPT es **PPO-ptx** (*pretraining mix*). Además de los gradientes de PPO, mezclan en cada minibatch gradientes de language modeling sobre D_{pretrain} , la distribución de texto crudo con la que se pre-entrenó GPT-3. El objetivo completo es

$$\text{objective}(\phi) = \mathbb{E}_{(x,y) \sim D_{\pi_\phi^{\text{RL}}}} \left[r_\phi(x, y) - \beta \log \frac{\pi_\phi^{\text{RL}}(y | x)}{\pi^{\text{SFT}}(y | x)} \right] + \gamma \mathbb{E}_{x \sim D_{\text{pretrain}}} [\log \pi_\phi^{\text{RL}}(x)], \quad (2b)$$

con $\gamma = 27.8$ y aproximadamente ocho veces más ejemplos de pretraining que episodios de RL por minibatch. El término $\gamma \mathbb{E}[\log \pi(x)]$ obliga al modelo a seguir siendo un buen language model de texto general mientras el RL lo alinea como asistente. En la práctica, PPO y PPO-ptx obtienen preferencias humanas similares en la API, pero solo PPO-ptx mitiga la regresión en tareas de lenguaje más amplias. Por eso el paper reserva el nombre *InstructGPT* para la variante con pretraining mix.

El entrenamiento RL corre 256 mil episodios sobre unos 31 mil prompts únicos, con batch de 512 y minibatches de 64. El RM y el critic permanecen fijos en 6B parámetros aunque la policy alcance 175B. PPO implementa, en el fondo, una estimación robusta del policy gradient

$$\nabla_\phi \text{objective} \propto \mathbb{E} \left[\sum_{t=1}^T \nabla_\phi \log \pi_\phi^{\text{RL}}(y_t | x, y_{<t}) \cdot \hat{A}_t \right],$$

con ventajas $\hat{A}_t$ de GAE y ratio clipped $\rho_t = \pi_\phi / \pi_{\phi_{\text{old}}}$. La correspondencia con el TAR es directa: la policy del LLM es $\pi(a | s)$ token a token; el reward del entorno es el score del RM (más la penalización KL por token); el critic sustituye al valor $V(s)$ del MDP, inicializado desde el RM.

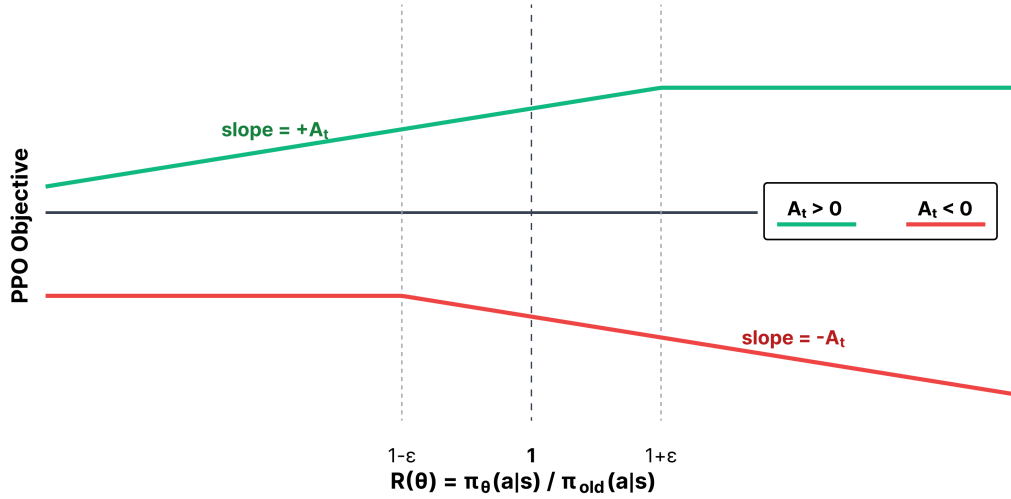


Figura 6: Loop PPO.

Resultados

Las evaluaciones humanas sobre la distribución de prompts de la API muestran una mejora progresiva en cada etapa del pipeline: GPT-3 base, GPT-3 con prompt cuidadoso, SFT, PPO y PPO-ptx. De forma llamativa, InstructGPT de 1.3B parámetros es preferido sobre GPT-3 de 175B en esa distribución; en comparaciones directas, la versión 175B de InstructGPT es elegida frente a GPT-3 175B en torno al 85 % de las veces. Eso no implica que el modelo pequeño “sepa más”, sino que el post-entrenamiento optimizó directamente lo que los evaluadores valoran en un asistente.

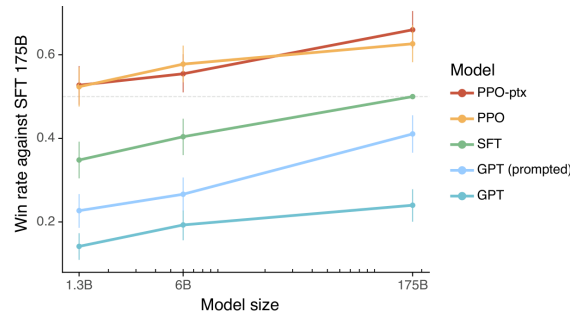


Figura 7: Evaluaciones humanas (Fig. 1).

Más allá de la preferencia global, los modelos PPO superan a GPT-3 en metadata recogida por los anotadores: siguen mejor las instrucciones explícitas, alucinan con menos frecuencia en tareas de dominio cerrado y producen respuestas más apropiadas en el rol de asistente.

Cierre

InstructGPT condensa, en un pipeline de tres etapas, la lógica del RLHF aplicado a LLMs. El SFT enseña por imitación y fija la referencia π^{SFT} . El Reward Model estima, vía Bradley-Terry, una función de utilidad cuyas diferencias reproducen las preferencias humanas — ecuación (1). PPO maximiza esa utilidad mientras penaliza la desviación de la referencia — ecuación (2a). PPO-ptx añade un término de language modeling sobre texto de pretraining para no sacrificar la capacidad lingüística general al pagar el *alignment tax* — ecuación (2b).

Desde el TAR, la lección operativa es que el LLM ya es la policy; el RM reemplaza la función de reward del entorno; PPO es el mismo algoritmo de trust region visto en el curso, adaptado a secuencias largas y rewards escasos. Lo específico de InstructGPT no es tanto el algoritmo base como la combinación de datos humanos moderados, un proxy de preferencias bien calibrado, regularización KL hacia el SFT, y

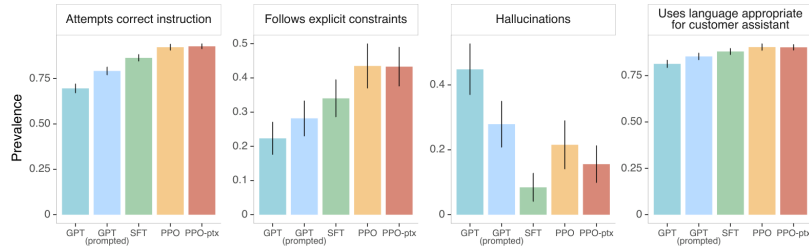


Figure 4: Metadata results on the API distribution. Note that, due to dataset sizes, these results are collapsed across model sizes. See Appendix E.2 for analysis that includes model size. Compared to GPT-3, the PPO models are more appropriate in the context of a customer assistant, are better at following explicit constraints in the instruction and attempting the correct instruction, and less likely

Figura 8: Metadata (Fig. 4).

la mezcla de pretraining que convierte un chatbot alineado en un asistente que sigue siendo, en buena medida, un language model general.

Referencias

- Ouyang et al. (2022). [Training language models to follow instructions with human feedback](#)
- Stiennon et al. (2020). [Learning to summarize with human feedback](#)
- Bradley, R. A.; Terry, M. E. (1952). Rank analysis of incomplete block designs.
- Lambert, N. [RLHF Book](#) · [Curso](#)