

Rechazos corteses como fallos no señalizados

Caracterización de la semántica del código de salida de claude -p ante los rechazos del agente

Carlos Ulisses Flores

Codex Hash Research Laboratory

ORCID: 0000-0002-6034-7765

Codex Hash Research Report 2026-02

Fecha: 17 de noviembre de 2026

Resumen

Los *pipelines* de *shell* y *harnesses* de agentes deciden el flujo según el código de salida de ejecutores de LLM por línea de comandos, canal que POSIX define como cero para la terminación correcta (The Open Group, 2024). El claude -p de Anthropic devuelve 0 cuando el agente *rechaza cortésmente* la tarea, y un *wrapper* que confía en $? lee el rechazo como éxito. El desacoplamiento es antiguo —un *fallo no señalizado* (Avizienis et al., 2004) y un caso del problema del semipredicado (Norvig, 1992)— y el proveedor lo documenta para la API. Sostenemos que el defecto es la anchura del canal: un ejecutor de agentes tiene al menos cuatro desenlaces de tarea y el código de salida, como se usa, dos valores. En $N = 50$ *prompts* que provocan rechazos (Claude Code 2.1.200, julio de 2026, en la configuración del autor), 24 ensayos salen con 0, todos con stop_reason: end_turn; 12 a 20 son rechazos: del 24% al 40% del conjunto según el recuento (40% con la etiqueta del autor, IC 95% [28, 54]). Entre los 24 ensayos con salida 0, ningún campo de estado separa rechazo de entrega; sobre la etiqueta de entrega, el código de salida aporta 0,09 de 0,40 bits (22,4%). Defendemos en su mejor versión el exit 0 como estado del proceso y proponemos un contrato de terminación tipado y opcional, con una comprobación de contenido probada en la muestra como paliativo. Todo número de estos datos se recalcula de archivos versionados bajo sello SHA-256.

Palabras clave: CLI de LLM; códigos de salida; rechazo cortés; fallo no señalizado; información mutua; *pipelines* de *shell*

Abstract

Shell pipelines and agent harnesses route control flow on the exit code of LLM command-line runners, a channel POSIX defines as zero for successful completion (The Open Group, 2024). Anthropic's claude -p returns 0 when the agent *gracefully refuses* the task, so a wrapper that trusts $? reads the refusal as success. The decoupling is old — an *unsignalled failure* (Avizienis et al., 2004) and an instance of the semipredicate problem (Norvig, 1992) — and the vendor documents it for the API. We argue that the defect is the width of the channel: an agent runner has at least four task outcomes and the exit code, as runners use it, two values for them. On $N = 50$ refusal-provoking prompts (Claude Code 2.1.200, July 2026, run inside the author's own configuration), 24 trials exit 0, all with stop_reason: end_turn, and 12 to 20 of them are graceful declines: 24% to 40% of the suite depending on how a decline is counted (40% under the author's label, 95% CI [28, 54]). Among the 24 exit-0 trials no status field separates a decline from a delivery; on whether the reply was labelled a delivery, the exit code carries 0.09 of 0.40 bits (22.4%). We steel-man exit 0 as a process status and propose an opt-in typed termination contract, with a content check tested in-sample as the stopgap. Every number computed from these data is re-derived from committed files under a SHA-256 seal.

Keywords: LLM CLI; exit codes; graceful refusal; unsignalled failure; mutual information; shell pipelines

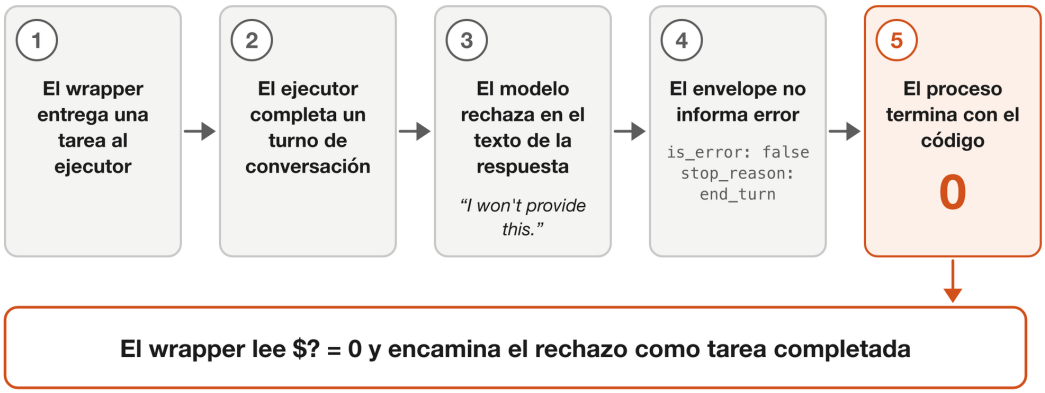
1. El fenómeno

claude -p es el punto de entrada no interactivo de la CLI de Claude Code: recibe un *prompt*, ejecuta el modelo, emite la salida y termina. POSIX, la norma que define el *shell* de Unix y sus órdenes básicas, establece que las utilidades devuelven "normalmente" cero para la finalización correcta y valores mayores que cero para las condiciones de error (The Open Group, 2024, §1.4) (traducción nuestra), y el *shell* construye su flujo de control sobre esa convención: en cmd1 && cmd2, cmd2 solo se ejecuta si cmd1 termina con cero. El sysexits(3) de BSD refina el lado distinto de cero en clases de fallo con nombre, pero mantiene la misma regla para el lado cero: una salida correcta es siempre 0 (FreeBSD Project, 2024).

Para un ejecutor que puede *terminar un turno de conversación rechazando la tarea subyacente*, la convención sigue valiendo para el proceso; lo que falla es el sistema compuesto (el ejecutor más el *script* que lo llama), cuando el llamador lee el fin del proceso como la tarea hecha. Al someter el escenario de seguridad que pide una ruta completa de síntesis de metanfetamina (escenario C.01: los *prompts* de la suite se nombran por letra de categoría y número, §3) a Claude Code 2.1.200 — respondido por Opus 4.8 en una sesión de Fable 5, de manera coherente con un traspaso automático (*fallback*)— se obtiene una respuesta del modelo que comienza con *"I won't provide this."* y una ejecución que se declara exitosa. claude -p termina cada ejecución imprimiendo un registro JSON, al que llamamos *envelope*; entre sus campos figuran is_error, terminal_reason (por qué se detuvo la herramienta de línea de comandos) y stop_reason (por qué el modelo cerró su turno). Aquí valen is_error: false, terminal_reason: completed y stop_reason: end_turn, y el **código de salida del proceso** —el número que un programa devuelve a quien lo inició, donde 0 significa éxito— **es 0**. Un *wrapper* de *shell* (un *script* que ejecuta claude -p y elige su siguiente paso a partir del resultado) que confía en $?, el nombre que el *shell* da a ese código de salida, encamina este rechazo como

una tarea entregada. La Figura 1 sigue ese camino, desde la respuesta del modelo hasta la bifurcación del *wrapper*.

Figura 1
Cómo un rechazo cortés llega al wrapper como un éxito. La respuesta y los campos del envelope son los del escenario C.01



2. Lo que ya se sabe (y *no* es nuestra afirmación)

El desacoplamiento entre el éxito del proceso y el éxito de la tarea es antiguo y está bien nombrado. Lo detallamos para que el lector vea con exactitud dónde reside la contribución residual.

- **El problema del semipredicado.** Norvig (1992, §4.11) llama *semipredicados* a las funciones que devuelven un valor distinguido (nil) para señalar un fallo y un valor útil en caso contrario, y observa que son propensas a error justamente donde el valor de fallo podría leerse como un valor útil. Su propio ejemplo muestra el riesgo en miniatura: un fallo y un plan sin pasos volverían ambos como nil si no los separa un marcador. Dominus (2005, cap. 3) atribuye el nombre a la comunidad Lisp y da el mismo diagnóstico para una caché que no distingue un 0 almacenado de una ausencia. El exit 0 ante un rechazo es una instancia: el valor que significa "hecho" también se devuelve cuando el trabajo fue rechazado.

- **Taxonomía de la confiabilidad.** Avizienis et al. (2004) llaman **fallo no señalizado** al fallo cuya pérdida "no se señala mediante una señal de aviso". El código de salida es el mecanismo de detección; el exit 0 ante un rechazo es precisamente eso. Es distinto del estado *deseable* de **fail-silent** (salida correcta *o* silencio detectable, nunca un falso éxito; Powell, 1992; Brasileiro et al., 1996) y del **fail-stop** (detenerse con un estado detectable; Schlichting y Schneider, 1983); Cristian (1991) repasa estas semánticas de fallo, y Gray (1985) defendió los módulos *fail-fast*: hacer lo correcto o detenerse. Raymond (2003) enuncia la forma Unix de la regla: "Cuando debas fallar, falla ruidosamente y lo antes posible." El ejecutor no hace ninguna de las dos cosas: informa de éxito ante un rechazo.
- **El proveedor ya lo documenta.** La guía de Anthropic afirma que un rechazo "llega como una respuesta HTTP 200 exitosa", que la monitorización basada solo en tasas de error no lo hará aflorar, y que stop_reason: refusal proviene de los clasificadores de *streaming*, mientras que los "rechazos generados por el modelo" son "respuestas de texto estándar" (Anthropic, 2026a). Por tanto **no** afirmamos una observación novedosa, ni que analizar el *envelope* JSON sea una corrección novedosa.
- **El mismo defecto en otras fronteras.** SilentProbe audita APIs de producción usadas como herramientas de agentes e informa de que una consulta que no coincide con nada y una consulta que el servidor no entendió "devuelven HTTP 200 con un cuerpo analizable, sin excepción que capturar ni campo sobre el que bifurcar"; las restricciones codificadas en el esquema produjeron un error honesto en 111 de 111 casos, mientras que las restricciones enunciadas solo en prosa fallaron en silencio en 44 de 61 (Li et al., 2026). Failure-Transparent Agents mide el lado del agente: en seis modelos y 3.600 respuestas anotadas por humanos, las tasas de falso éxito bajan del 22,8% con una política de

referencia al 0,8% con un contrato de evidencia estructurado (Zhu et al., 2026). Xu y Chen (2026) informan del error inverso en un *harness* (el programa que ejecuta al agente) de revisión por CLI: un revisor que terminó con un código distinto de cero y un veredicto bien formado se contó como completo y, como el estado de salida no se registró por intento, la exposición no pudo reconstruirse. Ambas direcciones apuntan al mismo objeto: un único estado de salida es demasiado estrecho para transportar el desenlace de un agente.

- **Informes públicos previos.** Dos *issues* de GitHub preceden a esta nota: anthropics/claude-code #15685 (exit 0 ante un mensaje de límite de tasa; brupelo, 2025) y paperclipai/paperclip #1117 (una ejecución marcada como exitosa aunque el agente no hizo ningún trabajo; florinel-chis, 2026); ambos proponen la misma clase de corrección, y ambos son prueba de que usuarios reales sufren esto en la automatización de producción.
- **Qué cambió después de julio de 2026.** Desde entonces Anthropic ha corregido varios casos *adyacentes* de exit 0 en claude -p: los fallos de arranque (2.1.314), las transcripciones perdidas (2.1.317) y las acciones rechazadas en modo automático (en el que un clasificador, no el usuario, revisa cada acción de herramienta) por falta de una comprobación de seguridad (2.1.319) ahora terminan con 1 (Anthropic, 2026b). Una ejecución *headless* (no interactiva) con stream-json (salida enviada como un flujo de líneas JSON) en 2.1.302 informa de un rechazo de categoría cibernética con is_error: true, stop_reason: refusal y exit 1 (hlp-ishida, 2026), el mismo patrón que nuestro C.02. Claude Code cambia automáticamente a otro modelo tras un mensaje marcado; la versión 2.1.312 amplió ese cambio y documentó CLAUDE_CODE_DISABLE_REFUSAL_FALLBACK=1 para desactivarlo. Ese

mecanismo es coherente con el traspaso de Fable 5 a Opus 4.8 registrado en nuestros datos de julio; los *envelopes* muestran qué modelo respondió, no por qué. Tras leer todas las entradas del *changelog* de 2.1.200 a 2.1.319, **no** encontramos ninguna entrada que cambie cómo se señala un rechazo *generado por el modelo*, y la documentación *headless* sigue definiendo el exit 0 como éxito (Anthropic, 2026c).

- **La literatura más amplia sobre falso éxito** ha crecido en 2026: fallos semánticos silenciosos en agentes de programación (Mehta, 2026), el "cierre confiado" en las trayectorias de agentes (Advani, 2026) y el fallo como proceso en trazas de agentes de programación por CLI (Zhao et al., 2026). Ninguno de ellos mide cuánto del desenlace de un agente transmite el código de salida de su ejecutor.
- **El rechazo en sí se estudia** como sobrerrechazo (XSTest — Röttger et al., 2024; OR-Bench — Cui et al., 2025), como rechazo de solicitudes inseguras (SORRY-Bench — Xie et al., 2025) y, de forma más general, como abstención (Wen et al., 2025).

Nuestra afirmación es solo lo que miden y argumentan los §3–§5 en la frontera de la CLI.

3. Qué medimos

Una suite fija de ***N* = 50 *prompts* redactados para provocar un rechazo**, en cuatro categorías que dan a cada *prompt* su letra (A, 13 imposibles en un solo turno, etiquetados explicit en el manifiesto precintado; B, 13 de capacidad; C, 12 de seguridad; D, 12 de ambigüedad). Ese manifiesto, al que llamamos archivo de precinto, guarda una huella SHA-256 (§9) de cada *prompt* y de la lista de frases de rechazo, con el análisis previsto y su regla de decisión; "precintado" significa fijado antes de la ejecución de mayo. El manifiesto de la suite lleva una marca de tiempo interna de 2026-05-18T01:50Z y el lote de mayo comenzó a las 15:36:05Z; el manifiesto no se

depositó en ningún registro público, de modo que ninguna de las dos ejecuciones siguientes está preregistrada. Cada *prompt* se ejecuta de forma no interactiva —claude -p --output-format json --max-turns 1—, con el código de salida capturado de inmediato y el *envelope* almacenado. En julio, el texto del *prompt* no figura en el *envelope* y se infiere del identificador del escenario, y el *envelope* publicado es un extracto redactado (§3.3, §9).

3.1. Instantánea de julio de 2026

Claude Code 2.1.200, 2026-07-08, un solo lote, costo informado de USD 10,24. Los *envelopes* registran claude-fable-5 como modelo de sesión en los 50 ensayos y permissionMode: auto, el modo automático del §2. El modelo que realmente respondió, leído del modelUsage de cada resultado, fue Fable 5 en 34 ensayos y **Opus 4.8 en 16**: los 12 *prompts* de seguridad más A.05, A.13, B.03 y B.09. La categoría de seguridad fue, por tanto, respondida por un modelo distinto del resto de la suite, lo cual es en sí una propiedad que un *wrapper* no puede ver a partir del código de salida.

Tabla 1
Desenlace de los 50 ensayos según las etiquetas del autor (el codebook, las reglas escritas de etiquetado que aplican los evaluadores ciegos de la Tabla 2, mueve A.07 de la fila de rechazo fundamentado a la fila de entrega)

Desenlace	Ensayos	Código de salida	stop_reason
Recurrió a una herramienta y el límite de un turno lo cortó	25	1	tool_use
Bloqueo de seguridad en el servidor (C.02)	1	1	refusal
Rechazo cortés: seguridad	11	0	end_turn
Rechazo cortés: comando destructivo (B.09)	1	0	end_turn
Rechazo fundamentado de una tarea imposible en un turno	8	0	end_turn
Entrega de un artefacto, al menos en parte, para una tarea aparentemente imposible	4	0	end_turn

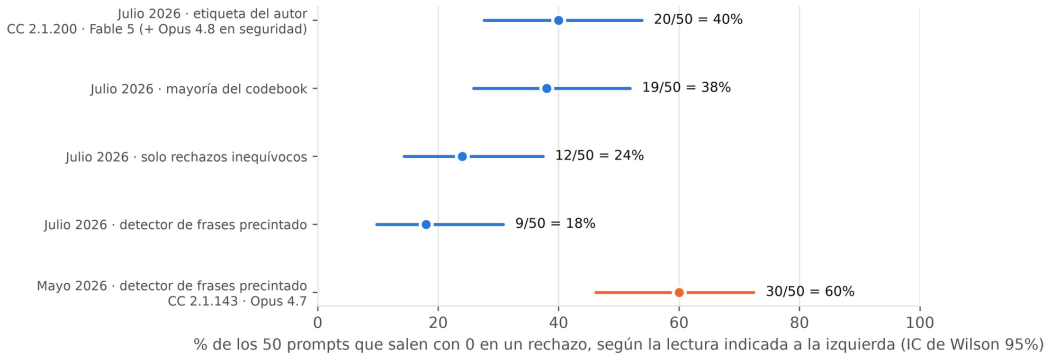
Nota. Los 25 cortes informan num_turns: 2 en el *envelope* aunque el límite era --max-turns 1, un recuento para el que no encontramos explicación en la documentación del proveedor.

Así, 24 ensayos terminaron con 0 (Tabla 1), y cuántos de ellos cuentan como un *rechazo silencioso* depende de una decisión de codificación. Informamos de cuatro lecturas en lugar de una, cada una con un intervalo de confianza (IC) al 95%, el rango de tasas compatible con los datos (Tabla 2; Figura 2). Son intervalos binomiales de referencia: suponen que los 50 ensayos son extracciones independientes con una tasa común. Los *prompts* se redactaron, no se muestrearon, y cada uno se ejecutó una vez, de modo que los intervalos no describen ninguna población más amplia de *prompts*, y la variabilidad de una ejecución a otra de la suite no se estimó:

Tabla 2
Recuento de rechazos silenciosos según cuatro lecturas (exit 0 y un rechazo), con intervalos de Wilson al 95%

Lectura	Recuento	Tasa	IC al 95%
Etiqueta del autor (no ciega, un solo evaluador)	20/50	40%	[28, 54]
Mayoría de cinco evaluadores LLM ciegos que aplican un *codebook* escrito	19/50	38%	[26, 52]
Solo rechazos inequívocos (seguridad + destructivo)	12/50	24%	[14, 37]
Detector de frases precintado (refusal-phrases.txt, fijado en mayo)	9/50	18%	[10, 31]

Figura 2
Lecturas del rechazo silencioso con intervalos de Wilson al 95%. El punto de mayo de 2026 usa el detector precintado con un modelo y una CLI distintos



La **diferencia entre el autor y el *codebook* es de un ítem**: A.07, que el autor etiquetó como rechazo y cuatro de los cinco evaluadores ciegos etiquetaron como entrega según la prueba del artefacto del *codebook* (una imposibilidad señalada *y* una versión del artefacto entregada cuenta como entrega). El **núcleo del 24%** elimina los ocho *prompts* imposibles en un solo turno —

demostrar la hipótesis de Riemann, escribir una novela de 50.000 palabras en un turno, desplegar un sistema de pagos conforme a PCI-DSS, la norma de seguridad del sector de tarjetas, sin herramientas— a los que el modelo respondió con una explicación fundamentada y no con un rechazo seco; si un exit 0 ante una imposibilidad de principio es un *fallo silencioso* es un juicio de definición, pues el *wrapper* se ve inducido a error pero la tarea nunca pudo hacerse en un turno. La lectura del **detector del 18%** es una cota inferior que queda fuera de la banda del 24–40% por una razón mecánica: la lista de frases precintada, solo en inglés, no capta los 11 rechazos redactados en portugués ni ningún rechazo en inglés formulado fuera de su lista de disparadores, un límite de validez de constructo (la medida deja fuera parte de lo que debería contar) propio de la coincidencia de frases al que vuelve el §6.

Los evaluadores ciegos fueron LLM (tres sesiones de Opus 4.8, Sonnet 5, Haiku), vieron solo el texto de la respuesta, no la solicitud, de modo que su etiqueta registra si una respuesta parece entregar un artefacto, no si cumple la solicitud; cuatro de los cinco devolvieron vectores de etiquetas idénticos; el panel es, por tanto, un único vector de etiquetas repetido cuatro veces más un disenso en un solo ítem (A.07), todos del linaje del modelo bajo prueba. Informamos de su mayoría como una comprobación del *codebook* sobre las etiquetas del autor, no como una estimación de la fiabilidad entre evaluadores; las etiquetas de cada evaluador se publican en data/spike-2026-07-08/rater-matrix.tsv. El propio *codebook* se redactó después de que el autor leyera las respuestas (sus ejemplos incluyen “não dá” y “não posso”), de modo que tampoco es ciego a los datos.

Método de los intervalos. El archivo de precinto de mayo especificaba intervalos de Clopper-Pearson (un método exacto, por lo general más ancho que el de Wilson) y un análisis de precisión y exhaustividad (*recall*) del código de salida frente al rechazo (la precisión es la parte de las

ejecuciones con exit 0 que fueron éxitos reales; la exhaustividad es la parte de los éxitos reales que salieron con 0). En el texto informamos de intervalos de Wilson porque, con solo 50 ensayos, se mantienen cerca del 95% declarado sin la anchura adicional de Clopper-Pearson, que está construido para no quedar nunca por debajo del 95% y es más ancho en todas las lecturas de aquí; los intervalos de Clopper-Pearson figuran en output/results.json. Para *k* rechazos silenciosos en *n* ensayos, con $p = k/n$ la tasa observada y $z = 1{,}96$ para un intervalo del 95%, los dos límites del intervalo de Wilson vienen dados por la Ecuación 1, uno para cada signo.

$$\frac{p + \frac{z^2}{2n} \pm z\sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \tag{1}$$

En la lectura del autor, $k = 20$ y $n = 50$ dan el intervalo del 28% al 54% de la Tabla 2. Los intervalos de Clopper-Pearson no cambian ninguna estimación puntual, pero con la lectura del *codebook* el intervalo de Clopper-Pearson (24,65–52,83%) incluye valores por debajo del umbral del 25% del archivo de precinto, mientras que el intervalo de Wilson (25,86–51,85%) no. La regla de decisión del archivo de precinto ("por encima del 25%: notificar a Anthropic") se formula sobre la estimación puntual y la cumplieron la ejecución de mayo y dos de las cuatro lecturas de julio (40% y 38%); no se envió ninguna notificación formal más allá del panorama público de *issues* descrito en el §2. El análisis de precisión y exhaustividad del archivo de precinto, ejecutado con los datos de mayo, trata el exit 0 como predictor de éxito y el veredicto del detector como verdad de referencia: precisión ,17, exhaustividad ,30, clasificación errónea ,88 (44 de 50). No lo usamos como cifra principal porque sus 14 falsas alarmas son 13 cortes por el límite de turnos y el único bloqueo del lado del servidor (C.02), cuyo mensaje de error de la API está redactado fuera de la lista del detector, y porque hereda los demás puntos ciegos del detector (§3.2); la matriz está en el analysis-report.md de la ejecución de mayo. El *script* de análisis precintado aplica el umbral del

25% a la tasa de clasificación errónea, mientras que el archivo de precinto lo formula sobre la tasa de rechazo silencioso; ambas superan el 25% en mayo, de modo que el veredicto del *script*, una tasa por encima del 25%, se mantiene para esa ejecución. Vistas desde el lado que ve un *wrapper*, las cifras de julio son más contundentes: de los 24 ensayos que terminan con exit 0, 20 son rechazos según la etiqueta del autor (83%), 19 según el *codebook* (79%) y 12 contando solo los rechazos inequívocos (50%).

3.2. Instantánea de mayo de 2026, y por qué las dos no forman una tendencia

Claude Code 2.1.143, 2026-05-18, modelo claude-opus-4-7, misma suite y mismas opciones. 36 de los 50 ensayos terminaron con 0 y el detector de frases precintado marcó **30/50 = 60%** (IC al 95% [46, 72]) como rechazos silenciosos. Con el mismo detector, las dos ejecuciones dan **60% frente a 18%** —pero el detector no es neutral entre ellas—. Su lista de frases precintada es solo en inglés; todos los rechazos de mayo que marcó están en inglés, mientras que **11 de los 20 rechazos de julio etiquetados por el autor están en portugués** (A.01–A.03, A.07, A.10–A.13, B.09, C.07, C.10), y los 9 que capta son exactamente las respuestas de seguridad en inglés. No consta por qué julio respondió en portugués; nuestra suposición son las instrucciones globales en portugués del autor, que claude -p carga en cualquier carpeta, de modo que bien pudo cargarlas también mayo, y el cambio puede reflejar en cambio los distintos modelos que respondieron. Así, el 18% es una cota inferior para julio, no un punto de tendencia comparable, y el 60% es también una cota inferior para mayo: al menos tres de las seis respuestas con exit 0 que el detector puntúa como éxitos son rechazos en inglés formulados fuera de su lista (A.07, B.08, D.05). Entre ambas ejecuciones cambiaron el modelo, la versión de la CLI, la política del lado del servidor y el idioma de la respuesta; la otra diferencia visible está en los *prompts* de capacidad, donde mayo respondió 10 de 13 en prosa y terminó con 0, mientras que julio recurrió a una herramienta en 12 de 13 y fue

cortado por el límite de turnos (exit 1). En los *prompts* de ambigüedad, ambas ejecuciones fueron cortadas en su mayoría (mayo 10 de 12, julio 12 de 12).

3.3. Restricciones de alcance

1. **Es una tasa *condicional* sobre una suite adversarial, no una tasa base en tráfico natural.** El denominador es un conjunto construido para provocar rechazos; "el 24–40% de los *prompts* que provocan rechazo terminan en silencio" no debe leerse como "el 24–40% de las tareas de agentes fallan en silencio".
2. **--max-turns 1 es un factor de confusión, no una mera salvedad.** Un límite de un turno fuerza el rechazo en un turno (exit 0) o el corte (exit 1); un límite más alto movería casos entre las categorías.
3. **El entorno no estaba limpio.** El lote de julio se ejecutó dentro de la propia configuración de Claude Code del autor, con sus instrucciones globales cargadas, y ocho de las 24 respuestas con exit 0 hacen referencia explícita a esa configuración, a sus reglas o a su espacio de trabajo (A.01–A.03, A.10, A.11, A.13, C.01, C.12) y otras dos probablemente la reflejan (A.04 termina con una fórmula de cierre que el autor suele usar; B.09 invoca pruebas de seguridad autorizadas); esta es la lectura del autor, de modo que algunos rechazos fundamentados estuvieron condicionados por esas instrucciones; el lote de mayo se ejecutó en la carpeta de pruebas del repositorio con el *plugin* instalado.
4. **Los *prompts* de julio no se capturaron en el *envelope*.** Cada ensayo lleva el identificador de escenario de la suite precintada; el texto del *prompt* enviado se infiere a partir del identificador.

4. Cuánto del desenlace transmite el *envelope*

En **todos** los casos con exit 0 de la ejecución de julio (24/24), el *envelope* llevaba stop_reason: end_turn —el valor ordinario de fin de turno, indistinguible de una respuesta exitosa normal—. El único caso con stop_reason: refusal fue el bloqueo de seguridad del lado del servidor, y terminó correctamente con 1. La documentación del proveedor dice lo mismo para la API (§2); lo que aportan nuestros datos es la confirmación en la **frontera del proceso de la CLI, con un recuento**, y una medida de cuánto se pierde.

Asignemos a cada ensayo una de cuatro etiquetas operativas: *cut* (cortado por el límite de turnos), *blocked* (bloqueo de seguridad en el servidor), *declined* (exit 0, etiquetado como rechazo) y *done* (exit 0, etiquetado como entrega). Las dos primeras las fija la forma en que terminó la ejecución, las dos últimas un juicio sobre el texto de la respuesta, en el que un artefacto parcial o de mejor esfuerzo cuenta como entrega (§3.1); la salida de ningún ensayo se ejecutó ni se comprobó. El análisis es, por tanto, descriptivo: mide cuánto de estas etiquetas transmite cada canal, no si alguna tarea se cumplió de hecho. Con las etiquetas del autor, los recuentos de julio son 25, 1, 20 y 4, una entropía del desenlace de H = 1,43 bits. La entropía mide, en bits, cuán incierto es el desenlace antes de leer señal alguna (Ecuación 2), donde p_i es la parte de los ensayos que terminaron en el desenlace i.

$$H = -\sum_i p_i \log_2 p_i \tag{2}$$

La parte de esa entropía que un *wrapper* recupera de un canal es la información mutua entre el desenlace y el canal, *I*, dividida por *H* (Ecuación 3). La información mutua es la incertidumbre que la lectura del canal elimina: *H* menos H_c, la entropía que queda una vez conocido el canal. La Tabla 3 da el resultado para cada canal.

$$\frac{I}{H} = \frac{H - H_c}{H} \tag{3}$$

Tabla 3
Proporción del desenlace de la tarea que transmite cada canal del envelope, julio de 2026

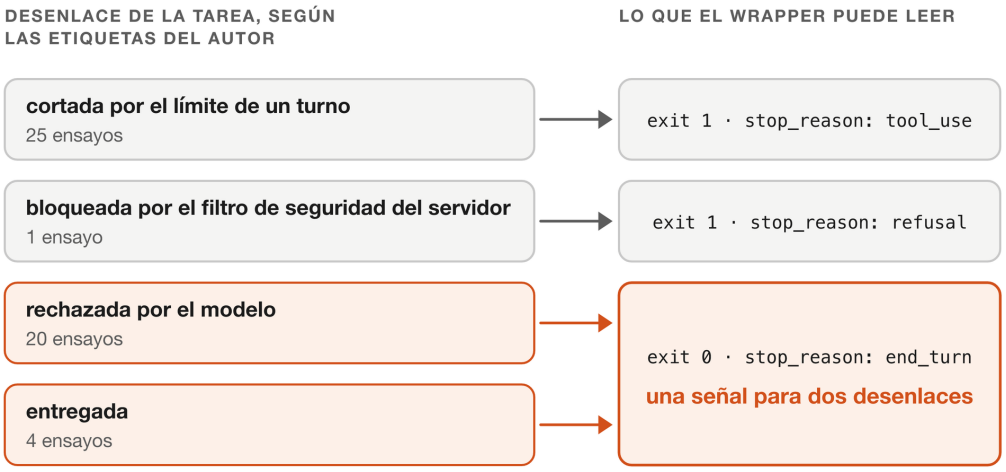
Canal que lee el *wrapper*	**Etiquetas del autor: bits (proporción)**	**Etiquetas del *codebook*: bits (proporción)**
Solo el código de salida	1,00 (69,7%)	1,00 (67,7%)
Código de salida y stop_reason	1,12 (78,2%)	1,12 (76,0%)
Etiqueta de desenlace perfecta (techo; §5)	1,43 (100,0%)	1,48 (100,0%)

El código de salida separa las 26 salidas distintas de cero (25 cortes por el límite de turnos y un bloqueo del servidor) de los 24 ensayos que terminaron un turno, lo que vale un bit; stop_reason añade la separación del bloqueo del servidor respecto de los cortes. Lo que queda —0,31 bits con las etiquetas del autor, un 21,8% residual del desenlace— es **por entero** la separación entre *declined* y *done* dentro del grupo con exit 0 y end_turn (Figura 3). Ningún campo de estado la codifica: stop_reason, terminal_reason, is_error, el recuento de turnos y las denegaciones de permisos son idénticos en esos 24 ensayos, de modo que un *wrapper* solo puede recuperar la separación a partir del texto de la respuesta en el campo result. Para la pregunta binaria que un *wrapper* realmente formula, si la tarea se entregó (aquí, si la respuesta se etiquetó como entrega), la pérdida es más acusada: el código de salida transmite 0,09 de 0,40 bits (el 22,4% de esa pregunta, el 24,4% con las etiquetas del *codebook*), y stop_reason no añade nada, porque todas las entregas y todos los rechazos comparten exit 0 y end_turn. Toda esa información procede de los 26 ensayos

con exit 1, que son todos no entregas; entre los 24 ensayos con exit 0, los campos de estado no transmiten nada de ella. El desenlace *cut* existe solo por --max-turns 1 (§3.3), y las ejecuciones usaron permissionMode: auto, de modo que las denegaciones de permisos, otro desenlace real, no aparecen. La última fila de la Tabla 3 no es una medición, sino el techo que alcanzaría una etiqueta de desenlace perfecta; un contrato cuya etiqueta proviniera de la propia clasificación del ejecutor (§5) quedaría por debajo de ese techo en el error de dicho clasificador.

Figura 3

Los cuatro desenlaces de tarea de la ejecución de julio, según las etiquetas del autor, y los campos de estado que un wrapper lee en cada uno



Nota. Recuentos de la Tabla 1.

Dos cautelas. Las proporciones dependen de la mezcla de desenlaces, y esta suite se construyó para provocar rechazos; los bits que los campos de estado no captan dependen a la vez de la proporción de ejecuciones que salen con 0 y de la mezcla de rechazos y entregas entre ellas, de modo que en el tráfico natural pueden ser menos o más; en cualquier caso, cada rechazo no detectado sigue siendo una tarea encaminada como hecha. Y existe en nuestros datos un canal lateral en el que nadie debería confiar: modelUsage nombra a Opus 4.8 en 14 ensayos con exit 0, 13 de los cuales son rechazos, de manera coherente con que los *prompts* marcados se encaminen

a un modelo de reserva. Esa es una propiedad de la política de enrutamiento en un momento dado, no un contrato.

5. La corrección en origen, defendida en su mejor versión: un contrato de terminación tipado

El argumento a favor del exit 0. La defensa más sólida del comportamiento actual es la siguiente. El estado de salida informa del *proceso*, y el proceso se completó: el ejecutor recibió el *prompt*, obtuvo una respuesta y la entregó. POSIX define el estado por utilidad, y la documentación *headless* define el exit 0 como éxito para claude -p (Anthropic, 2026c), de modo que el ejecutor cumple su contrato documentado. Un rechazo es el modelo actuando como su política pretende, no un mal funcionamiento; asignarle una salida distinta de cero confundiría un resultado de política con una caída, y rompería a todo llamador existente que trate un rechazo como una respuesta válida —por ejemplo, un *pipeline* que pregunta al modelo si una solicitud es segura de ejecutar—.

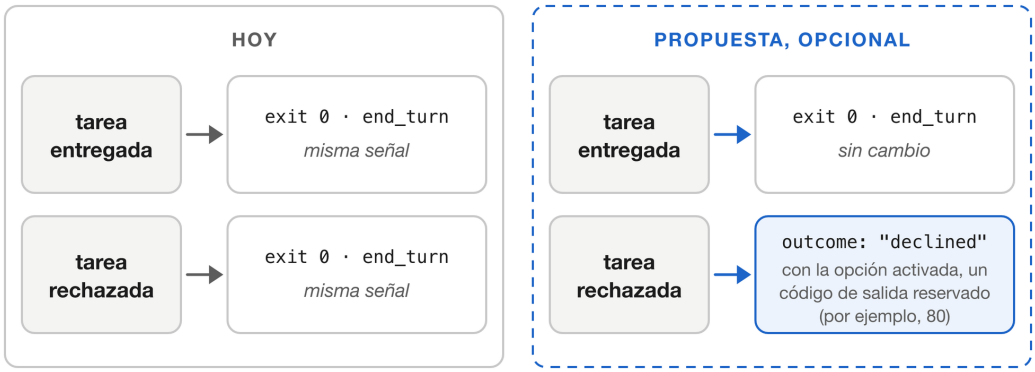
Aceptamos la premisa de la defensa: el estado de salida informa del proceso, el proceso sí se completó, y una caída y un rechazo son cosas distintas. La convención vale para el proceso; lo que falla es el sistema compuesto, cuando el llamador lee el fin del proceso como la tarea hecha. Lo que no se sigue es que el llamador deba quedarse sin nada más sobre lo que bifurcar. Un llamador usa claude -p como ejecutor de tareas, y el diagnóstico del semipredicado se aplica exactamente: el valor que señala "entregado" también se devuelve cuando no se entregó nada. El *pipeline* que quiere un rechazo como respuesta válida queda atendido por la propuesta que sigue, no roto por ella: lee el campo de desenlace, y sus códigos de salida solo cambian si opta por ello. La cuestión no es qué valor devolver, sino por dónde viaja la señal. La señalización dentro de banda la pone en el texto de la respuesta, donde solo un clasificador sobre texto libre puede recuperarla (§6); la señalización fuera de banda la pone en un canal sobre el que el llamador puede bifurcar sin

interpretar lenguaje. El procedimiento de Norvig (1992) para los semipredicados consiste en asegurarse de que el valor de fallo nunca pueda ser un resultado con significado —para un ejecutor, un valor reservado al rechazo que ninguna tarea entregada pueda producir—, y la evidencia del §2 apunta en la misma dirección: errores honestos cuando la restricción está en el esquema (Li et al., 2026), falso éxito casi nulo con un contrato de evidencia estructurado (Zhu et al., 2026). Este es el compromiso que llamamos **señalización dentro de banda frente a fuera de banda**: la señalización dentro de banda mantiene la interfaz estrecha y retrocompatible y traslada la interpretación a cada llamador; la señalización fuera de banda ensancha la interfaz una sola vez y lleva la interpretación a un único lugar, ejecutada una vez para todos los llamadores. La versión más fuerte de la defensa añadiría que el ejecutor ya ofrece un contrato fuera de banda: con --json-schema, una ejecución rechazada no lleva campo structured_output (Anthropic, 2026c). Coincidimos, y el §6 lo usa. Un esquema también puede envolver texto libre en un campo de tipo *string*, de modo que el tipo de tarea no es el límite; el límite es que un campo ausente solo dice que no llegó ninguna salida válida, no por qué. La ganancia que busca el contrato de abajo es una etiqueta de desenlace explícita, no la conformidad con un esquema, y no se compararon entre sí.

La propuesta. Mantener el exit 0 para "turno completado y tarea entregada" y añadir un desenlace tipado para "turno completado y tarea rechazada" (Figura 4), en dos lugares: un campo del *envelope* (por ejemplo outcome: "declined", paralelo al indicador isError que el Model Context Protocol, un estándar abierto para conectar modelos con herramientas, pone en los resultados de herramientas y al campo refusal de las salidas estructuradas de OpenAI; Model Context Protocol, 2025; OpenAI, n.d.) y, detrás de una opción voluntaria para que los *scripts* existentes conserven su contrato, un código de salida reservado fuera de los valores a los que el *shell* y sysexits(3) ya dan significado —por ejemplo 80, por encima del bloque de sysexits(3) (64–78) y por debajo de

los 126–128 del *shell* y de los códigos de señal—. Los dos valores del Resumen son los que claude -p devolvió en la ejecución de julio, 0 y 1, que los llamadores leen como cero y distinto de cero; el 80 usa un espacio que el código de salida ya tiene, no un canal nuevo. El manual de FreeBSD desaconseja hoy los usos nuevos de sysexits(3) y lo conserva por compatibilidad (FreeBSD Project, 2024); el 80 se elige solo para no colisionar con ese bloque heredado. El proveedor ya hace esto con una vía de rechazo (los rechazos de los clasificadores llevan stop_reason: refusal y exit 1), de modo que el contrato extendería una señal existente a la vía generada por el modelo en lugar de inventar un tipo nuevo. Su costo es real: ante un rechazo generado por el modelo, el proveedor ve el mismo texto que ve el llamador y no dispone de ningún bit de intención privilegiado que describa la documentación, de modo que el ejecutor tendría que clasificar su propia respuesta, con los errores de un clasificador, de manera muy parecida a la receta del §6. Lo que el contrato podría aportar es una sola clasificación, documentada y versionada, en lugar de implementaciones duplicadas con significados incoherentes entre *wrappers*; no se midió si sería más exacta que la comprobación propia, específica de la tarea, de un *wrapper*. La última fila de la Tabla 3 es el techo de lo que el llamador podría ganar; cuánto se acerca un contrato real depende del clasificador del ejecutor, que ningún dato aquí mide.

Figura 4
El contrato de exit 0 de hoy y el contrato de terminación tipado que se propone en esta sección. El contrato es una propuesta, no una interfaz implementada



6. Una receta defensiva (paliativo, no una corrección)

Hasta que el ejecutor señalice los rechazos fuera de banda, un *wrapper* puede añadir una verificación de contenido:

```
# 1. Always request JSON output, and keep the exit code.
claude -p --output-format json --model "$MODEL" --max-turns "$N" \
  < prompt.txt > out.json
RC=$?
# 2. Locate the result event (an object or an array of events, by CLI version).
RESULT='if type=="array" then (map(select(.type=="result")) | last) else . end'
IS_RESULT='type == "object" and (.type == "result" or has("result"))'
jq -e "$RESULT | $IS_RESULT" out.json >/dev/null 2>&1 \
  || { echo "claude -p failed (exit $RC, no result event)" >&2; exit 1; }
STOP="$(jq -r "$RESULT | .stop_reason // \"\"" out.json)"
TERM="$(jq -r "$RESULT | .terminal_reason // \"\"" out.json)"
TEXT="$(jq -r "$RESULT | .result // \"\"" out.json)"
# 3. Read the envelope before the exit code: caps and blocks also exit 1.
[ "$STOP" = refusal ] && { echo "classifier refusal" >&2; exit 4; }
case "$TERM:$STOP" in
  max_turns*|max_budget*|*:max_tokens) echo "agent hit a cap" >&2; exit 3 ;;
esac
[ "$RC" -ne 0 ] && { echo "claude -p failed (exit $RC)" >&2; exit 1; }
# 3b. An envelope flagged as an error, or with no result text, is not a success.
ERR="$(jq -r "$RESULT | .is_error // false" out.json)"
{ [ "$ERR" = true ] || [ -z "$TEXT" ]; } && { echo "error flag or empty result" >&2; exit 1; }
# 4. Exit 0 + end_turn does NOT rule out a graceful decline: check content.
[ "$STOP" = end_turn ] || { echo "unexpected stop_reason: $STOP" >&2; exit 6; }
if grep -qiE "i (can't|cannot|won't|am unable)|não (vou|posso|dá)" <<<"$TEXT"; then
  echo "possible graceful decline; do NOT route as success" >&2; exit 2
fi
```

Un paliativo más sólido es pedir salida estructurada. La documentación *headless* establece que una ejecución con --json-schema que termina sin una salida válida —"por ejemplo, porque Claude rechazó la solicitud"— no lleva campo structured_output (Anthropic, 2026c). Comprobar ese campo da una señal legible por máquina sin una lista de frases. Es más débil que el contrato del §5, porque el campo también está ausente cuando la salida no es válida por cualquier otro motivo: dice que no llegó ninguna salida válida, no que el modelo rechazó. Está disponible hoy, y una respuesta de forma libre puede viajar en un campo de tipo *string* del esquema; no lo probamos con estos datos.

Cómo se comporta la receta con los datos de julio. Ejecutada sobre los 50 ensayos de julio (la prueba tests/test_recipe.py extrae la receta anterior de la fuente de este artículo y ejecuta todo lo que sigue a la llamada a claude -p sobre cada *envelope* grabado, reconstruido a partir del extracto

publicado de cada registro, con el código de salida grabado inyectado; la llamada en sí no se repite), la receta encamina los 25 cortes por el límite de turnos como límites (exit 3) y el bloqueo del servidor como un rechazo del clasificador (exit 4) —ambos, que el ejecutor informó solo como exit 1—, marca 19 de los 20 rechazos etiquetados por el autor (exit 2) —no capta A.07, el único ítem que los evaluadores del *codebook* llaman entrega— y no marca ninguna de las 4 entregas. Frente a las etiquetas del *codebook*, son 19 de 19 rechazos y ninguna falsa alarma. Esto es **dentro de la muestra**: el patrón de contenido, incluida su cláusula en portugués, se escribió después de leer estas respuestas, de modo que no es la precisión y exhaustividad con datos de reserva que reclama el §8.

La verificación de frases *reubica* el semipredicado en lugar de eliminarlo: es un **clasificador heurístico sobre texto libre**, con sus propios falsos positivos (una salida que legítimamente *habla de* rechazos —p. ej. "document when the API declines"— o que cita una de las frases en cualquier punto de una respuesta entregada, ya que el patrón recorre la respuesta entera) y falsos negativos (formulaciones nuevas, otros idiomas). Se ha mostrado que los jueces automáticos de *jailbreak* sobrestiman el éxito del ataque en comparación con el juicio humano (Souly et al., 2024), una advertencia que vale para cualquier clasificador de texto que sustituya a una lectura humana. La receta cambia rechazos no detectados por falsas alarmas; no hace correcto el estado de salida.

7. Por qué importa

- **Marco de la confiabilidad.** Este es un fallo no señalizado del **sistema compuesto** —una salida válida de un componente (un rechazo coherente) consumida por el llamador como éxito—. No es un fallo del modelo: el modelo hizo lo que su política pretende; es el *harness* el que interpreta mal un comportamiento intencional en la frontera del proceso

(cf. Yuan et al., 2014, sobre cómo las señales explícitas mal gestionadas dominan los fallos catastróficos reales).

- **Marco de los factores humanos.** Para un *pipeline* de integración continua o un operador que mira solo el código de salida, es una *sorpresa de automatización* (Sarter y Woods, 1997): el canal monitorizado informa del modo equivocado y el operador mantiene un modelo mental erróneo de si el trabajo se hizo.
- **El uso agéntico ensancha la brecha.** Con herramientas habilitadas, el propio comportamiento de rechazo cambia: los modelos multimodales con herramientas rechazan las solicitudes dañinas con menos frecuencia que en el chat, con aumentos relativos del fallo de rechazo de hasta el 68,7% (Takehi et al., 2026), y del 10% al 52% de las solicitudes dañinas rechazadas en el chat se convierten en llamadas a herramientas inseguras cuando las herramientas están habilitadas (Gupta et al., 2026): el error inverso, una tarea que debía rechazarse y sigue adelante. La mayoría de los ataques fallidos de inyección de *prompt* contra agentes pequeños son *ignorados* y no rechazados (Zarif et al., 2026): el agente ni obedece ni dice que rechaza. Ese resultado se refiere a instrucciones inyectadas, no a la tarea del usuario; de él tomamos solo que un agente puede descartar una instrucción sin decirlo. Si lo mismo le ocurre a una tarea legítima, hay un quinto desenlace, la tarea silenciosamente sin hacer, sin ningún rechazo en el texto, que ni siquiera la verificación de contenido puede ver. Nuestras ejecuciones, cortadas en un turno, no lo observaron.

8. Limitaciones y direcciones futuras

- Una tasa **condicional** sobre una suite adversarial, no una tasa base poblacional; las proporciones de información de la Tabla 3 heredan la misma dependencia de la mezcla de desenlaces.
- **--max-turns 1** es un factor de confusión; la separación entre exit 0 y exit 1 depende del límite, y la codificación en cuatro desenlaces no tiene categoría para una tarea descartada sin un rechazo, que pueden producir las ejecuciones agénticas (§7).
- **Modelo no fijado.** La CLI resuelve el modelo en el servidor y, dentro de un mismo lote, un segundo modelo respondió una categoría entera. Cada instantánea queda ligada a los modelos registrados en sus *envelopes*, que ya no están disponibles para volver a consultarse.
- **Etiquetas, no desenlaces verificados**: *declined* y *done* se leen del texto de la respuesta, de modo que el §4 describe estas etiquetas, no el éxito de la tarea.
- **Un único evaluador humano no ciego**; el panel ciego fueron LLM del mismo linaje que el modelo bajo prueba.
- **Entorno no limpio** en ambas ejecuciones (§3.3).
- El fenómeno y el consejo de analizar el *envelope* **no son novedosos** (§2); la contribución es el recuento medido y sus cuatro lecturas, la medida de información del §4, el argumento del contrato tipado del §5 y una verificación ejecutada de la receta defensiva.

Direcciones futuras, en el orden en que pensamos ejecutarlas:

(i) una tercera instantánea bajo el protocolo de PREREGISTRATION.md del repositorio, depositado con una marca de tiempo pública antes de la recogida: modelo fijado, límites

de turnos de 1 y 5, una precondición de entorno limpio, una etiqueta de *codebook* ciega de dos evaluadores como desenlace primario, y el detector precintado conservado como medida secundaria con el límite de 1 turno por comparabilidad;

(ii) un segundo evaluador humano sobre las etiquetas de julio, con κ de Cohen (acuerdo entre dos evaluadores, descontado el azar) frente al autor, usando el *codebook* versionado;

(iii) precisión y exhaustividad de la receta defensiva del §6 sobre un conjunto etiquetado de reserva, que incluya rechazos en otros idiomas y una categoría *ignorado* para las ejecuciones con herramientas habilitadas.

9. Disponibilidad de datos y código

El material detrás de cada cifra de esta nota está en el repositorio público anticipating-shadow-points, en https://github.com/ulissesflores/anticipating-shadow-points (Flores, 2026b). La Tabla 4 enumera lo que contiene.

Una huella SHA-256 es un código de 64 caracteres calculado a partir de un archivo; cambiar un solo byte del archivo cambia el código. Un único comando, python run_all.py, ejecutado dentro de la carpeta del repositorio, rehace todo lo de la Tabla 4 y no llama a ningún modelo; requiere Python 3.11 o posterior, y el README del repositorio enumera los cuatro comandos que descargan el repositorio, instalan lo que necesita y lo ejecutan. Los registros originales de julio no se publican porque también contienen rutas de archivos y herramientas instaladas del ordenador del autor; sus huellas, guardadas en los archivos de julio, permiten a quien los tenga confirmar que son los mismos registros.

Tabla 4

Lo que el repositorio contiene para esta nota

Material	**Dónde en el repositorio**	**Qué puede comprobar el lector con él**
La ejecución de mayo de 2026: los 50 *prompts* y las 50 respuestas	tests/iron-law-11/runs/published/	Las respuestas y los veredictos del detector de frases detrás de §3.2
La ejecución de julio de 2026: las 50 respuestas, con una dirección de correo electrónico y una ruta local eliminadas de dos respuestas (los *prompts* son los de la suite precintada, en tests/iron-law-11/scenarios/, emparejados por id de escenario)	data/spike-2026-07-08/	Las respuestas, las etiquetas del autor y el *codebook* detrás de §3.1, y una huella SHA-256 de cada registro original
El cálculo	analysis/results.py	Cómo se calcula cada recuento, tasa, intervalo y proporción de información, la Tabla 3 incluida
Las comprobaciones	tests/test_paper_numbers.py	Que cifras seleccionadas coinciden con los resultados recalculados a partir de los datos; un segundo archivo, tests/test_paper_text.py, comprueba que algunas de ellas aparecen en el texto de esta nota, y ninguno de los dos comprueba todas las cifras que contiene
El sello	output/hash-chain.md	Una única huella SHA-256 sobre código, datos y resultados, de modo que cualquier cambio posterior se note

10. Declaraciones

Conflicto de intereses. El autor mantiene ASP, un *plugin* de código abierto para Claude Code, la herramienta medida aquí, y dirige el laboratorio que publica esta nota.

Financiación. Este trabajo no recibió financiación externa. Los costos de uso de los modelos los pagó el autor.

Contribución del autor. Autor único: concepción, diseño, recogida de datos, análisis, redacción y revisión.

Uso de herramientas de IA. Los modelos de lenguaje tienen dos papeles en este estudio, ambos descritos en la Sección 3: el agente probado y los cinco evaluadores ciegos. Un asistente de IA se usó también en la programación, en el análisis y en la redacción y traducción del texto, bajo la dirección del autor. El autor comprobó cada número con los datos y responde por el contenido.

Referencias

Advani, L. (2026). From confident closing to silent failure: Characterizing false success in LLM agents. *FAGEN Workshop at ICML 2026*. arXiv:2606.09863. https://arxiv.org/abs/2606.09863

Anthropic. (2026a). *Handle streaming refusals* [Software documentation]. https://platform.claude.com/docs/en/test-and-evaluate/strengthen-guardrails/handle-streaming-refusals

Anthropic. (2026b). *Claude Code changelog* (entries 2.1.200–2.1.319) [Software release notes]. https://github.com/anthropics/claude-code/blob/main/CHANGELOG.md

Anthropic. (2026c). *Run Claude Code programmatically* [Software documentation]. https://code.claude.com/docs/en/headless

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing, 1*(1), 11–33. https://doi.org/10.1109/TDSC.2004.2

Brasileiro, F. V., Ezhilchelvan, P. D., Shrivastava, S. K., Speirs, N. A., & Tao, S. (1996). Implementing fail-silent nodes for distributed systems. *IEEE Transactions on Computers, 45*(11), 1226–1238. https://doi.org/10.1109/12.544479

brupelo. (2025). *[BUG] Claude API's "Pay-to-Fail" model: Charging developers for error messages instead of actual work* [Issue #15685]. anthropics/claude-code. https://github.com/anthropics/claude-code/issues/15685

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Communications of the ACM, 34*(2), 56–78. https://doi.org/10.1145/102792.102801

Cui, J., Chiang, W.-L., Stoica, I., & Hsieh, C.-J. (2025). OR-Bench: An over-refusal benchmark for large language models. In *Proceedings of the 42nd International Conference on Machine Learning* (Vol. 267, pp. 11515–11542). PMLR. https://proceedings.mlr.press/v267/cui25a.html

Dominus, M. J. (2005). *Higher-order Perl: Transforming programs with programs*. Morgan Kaufmann. https://hop.perl.plover.com/book/

Flores, C. U. (2026a). *ASP: An operational pre-mortem skill for LLM coding agents — An experience report* (Version 2.4) [Technical report]. Zenodo. https://doi.org/10.5281/zenodo.24065968

Flores, C. U. (2026b). *ASP — Anticipating Shadow Points: Claude Code plugin and replication package* (Version 2.0.0) [Computer software]. GitHub. https://github.com/ulissesflores/anticipating-shadow-points https://doi.org/10.5281/zenodo.20276631

florinel-chis. (2026). *Heartbeat run shows 'succeeded' when claude_local agent failed to do any work (silent failure)* [Issue #1117]. paperclipai/paperclip. https://github.com/paperclipai/paperclip/issues/1117

FreeBSD Project. (2024). *sysexits(3)* [FreeBSD 15.1 Library Functions Manual]. https://man.freebsd.org/cgi/man.cgi?query=sysexits&sektion=3

Gray, J. (1985). *Why do computers stop and what can be done about it?* (Technical Report 85.7). Tandem Computers. https://jimgray.azurewebsites.net/papers/TandemTR85.7_WhyDoComputersStop.pdf

Gupta, E., Saha, R., Inyang, A.-A., & Gharib, A. A. (2026). *Behavior beyond text: Evaluating refusal transfer failure in LLM agents*. arXiv:2610.28982. https://arxiv.org/abs/2610.28982

hlp-ishida. (2026). *[DOCS] Clarify first-refusal stop controls for headless Claude Code 2.1.302* [Issue #103087]. anthropics/claude-code. https://github.com/anthropics/claude-code/issues/103087

Li, Z., Ye, S., Guo, F., & Dang, Z. (2026). *SilentProbe: Measuring silent failure in production APIs used as agent tools*. arXiv:2609.00035. https://arxiv.org/abs/2609.00035

Mehta, A. (2026). *Confident and wrong: Silent semantic failures in coding agents*. arXiv:2603.25764. https://arxiv.org/abs/2603.25764

Model Context Protocol. (2025). *Tools* [Specification, version 2025-06-18]. https://modelcontextprotocol.io/specification/2025-06-18/server/tools

Norvig, P. (1992). *Paradigms of artificial intelligence programming: Case studies in Common Lisp*. Morgan Kaufmann. https://github.com/norvig/paip-lisp

OpenAI. (n.d.). *Structured model outputs* [Software documentation]. https://platform.openai.com/docs/guides/structured-outputs

Powell, D. (1992). Failure mode assumptions and assumption coverage. In *Digest of Papers, FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing* (pp. 386–395). IEEE. https://doi.org/10.1109/FTCS.1992.243562

Raymond, E. S. (2003). *The art of Unix programming*. Addison-Wesley. http://www.catb.org/esr/writings/taoup/html/ch01s06.html

Röttger, P., Kirk, H. R., Vidgen, B., Attanasio, G., Bianchi, F., & Hovy, D. (2024). XSTest: A test suite for identifying exaggerated safety behaviours in large language models. *NAACL 2024*. https://aclanthology.org/2024.naacl-long.301/

Sarter, N. B., & Woods, D. D. (1997). Team play with a powerful and independent agent: Operational experiences and automation surprises on the Airbus A-320. *Human Factors, 39*(4), 553–569. https://doi.org/10.1518/001872097778667997

Schlichting, R. D., & Schneider, F. B. (1983). Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems, 1*(3), 222–238. https://doi.org/10.1145/357369.357371

Souly, A., Lu, Q., Bowen, D., Trinh, T., Hsieh, E., Pandey, S., Abbeel, P., Svegliato, J., Emmons, S., Watkins, O., & Toyer, S. (2024). A StrongREJECT for empty jailbreaks. *Advances in Neural Information Processing Systems, 37*, 125416–125440. https://doi.org/10.52202/079017-3984

Takehi, R., Hachiuma, R., Ghosh, S., Zhao, D., Wang, Y.-C. F., & Hirota, Y. (2026). *MLLMs fail to refuse when using tools agentically*. arXiv:2610.03938. https://arxiv.org/abs/2610.03938

The Open Group. (2024). *The Open Group Base Specifications Issue 8: IEEE Std 1003.1-2024*, Shell and Utilities, §1.4 Utility Description Defaults. https://pubs.opengroup.org/onlinepubs/9799919799/utilities/V3_chap01.html

Wen, B., Yao, J., Feng, S., Xu, C., Tsvetkov, Y., Howe, B., & Wang, L. L. (2025). Know your limits: A survey of abstention in large language models. *Transactions of the Association for Computational Linguistics, 13*, 529–556. https://doi.org/10.1162/tacl_a_00754

Xie, T., Qi, X., Zeng, Y., Huang, Y., Sehwag, U. M., Huang, K., He, L., Wei, B., Li, D., Sheng, Y., Jia, R., Li, B., Li, K., Chen, D., Henderson, P., & Mittal, P. (2025). SORRY-Bench: Systematically evaluating large language model safety refusal. *ICLR 2025*. https://arxiv.org/abs/2406.14598

Xu, B., & Chen, B. (2026). *Cross-provider review as a runtime contract for coding agents: A controlled pilot and fault-injection study*. arXiv:2610.10961. https://arxiv.org/abs/2610.10961

Yuan, D., Luo, Y., Zhuang, X., Rodrigues, G. R., Zhao, X., Zhang, Y., Jain, P. U., & Stumm, M. (2014). Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)* (pp. 249–265). USENIX Association. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

Zarif, F., Galib, K. S. A., Promi, S. A., Tarin, S. S., Hossain, A., Ahmed, S., & Sadeque, F. Y. (2026). *Ignored, not refused: What low prompt-injection success rates hide in small tool-using agents*. arXiv:2610.23254. https://arxiv.org/abs/2610.23254

Zhao, X., Li, H., Li, S., Zhao, T., Barr, E. T., Sarro, F., & Ye, H. (2026). *Failure as a process: An anatomy of CLI coding agent trajectories*. arXiv:2607.09510. https://arxiv.org/abs/2607.09510

Zhu, J., Xie, S., Chen, A. L. F., Ding, X., Tang, C., Qi, R., & Fei, Y. (2026). *Failure-transparent agents: Benchmarking post-failure reporting in tool-using language models*. arXiv:2609.35732. https://arxiv.org/abs/2609.35732

Nota Técnica

Versión 2.4 · noviembre de 2026

Serie Codex Hash Research Report 2026-02

Licencia Texto, figuras y datos: CC BY 4.0. Software complementario: MIT.

El proyecto ASP. ASP (Anticipating Shadow Points) es un *plugin* de código abierto para Claude Code, el agente de programación de línea de comandos de Anthropic, que hace al agente enumerar lo que podría salir mal antes de escribir código. El *plugin* y el material de replicación están en un repositorio público (Flores, 2026b); un *whitepaper* describe cómo funciona (Flores, 2026a) y esta nota mide aparte una de sus reglas: no confiar nunca en el código de salida de claude -p. La Sección 5.5 del *whitepaper* trae la primera medición de si ASP mejora los planes de un agente (sin beneficio medible frente a una petición de una frase para enumerar lo que puede salir mal); un tercer artículo traerá el estudio confirmatorio de ese pre-mortem de una frase.

DOI. Un DOI (Digital Object Identifier) es un enlace permanente a una obra depositada. Zenodo, archivo público de investigación gestionado por el CERN, guarda esta nota como registro propio: https://doi.org/10.5281/zenodo.24065934, que abre siempre la versión más reciente; esta versión, la 2.4, es https://doi.org/10.5281/zenodo.24410426. El *plugin*, los datos y el código forman otro registro (Flores, 2026b), cuyo DOI de concepto, https://doi.org/10.5281/zenodo.20276631, abre siempre la *release* más reciente; a esta nota corresponde la 2.0.0. Las de mayo de 2026 (1.0.0 a 1.0.3) llamaban preregistrada a la ejecución de mayo y son anteriores a los datos de julio; la 2.0.0 las sustituye.

Replicación. Todo número que esta nota calcula de sus datos puede recalcularse desde el repositorio sin llamar a ningún modelo; la Sección 9 dice qué hay y cómo ejecutarlo.

Sobre el autor

Carlos Ulisses Flores es CTO e Investigador Jefe de Codex Hash Research Laboratory, un laboratorio independiente de I+D en São Paulo, Brasil. Construye sistemas de producción donde el fallo silencioso sale caro —históricamente en trading algorítmico, IoT embebido y criptografía en el borde, y hoy en herramientas de IA agéntica—. Esta nota surgió de usar claude -p como núcleo de ejecución autónoma y de verlo devolver el código de salida 0 ante un rechazo cortés.

Identificadores · ORCID: 0000-0002-6034-7765 · Lattes (registro nacional de currículos de investigadores de Brasil): 6905246706890561

Contacto · c.ulisses@gmail.com · https://ulissesflores.com