

Rifiuti garbati come fallimenti non segnalati

Caratterizzazione della semantica del codice di uscita di claude -p nei rifiuti dell'agente

Carlos Ulisses Flores

Codex Hash Research Laboratory

ORCID: 0000-0002-6034-7765

Codex Hash Research Report 2026-02

Data: 17 novembre 2026

Riassunto

Le *pipeline* di *shell* e gli *harness* di agenti decidono il flusso in base al codice di uscita degli esecutori di LLM da riga di comando, canale che POSIX definisce come zero per la terminazione corretta (The Open Group, 2024). Il claude -p di Anthropic restituisce 0 quando l'agente *rifiuta garbatamente* il compito, e un *wrapper* che si fida di $? legge il rifiuto come successo. Il disaccoppiamento è antico — un *fallimento non segnalato* (Avizienis et al., 2004) e un caso del problema del semipredicato (Norvig, 1992) — e il fornitore lo documenta per l'API. Sosteniamo che il difetto sia l'ampiezza del canale: un esecutore di agenti ha almeno quattro esiti del compito e il codice di uscita, come viene usato, due valori. Su $N = 50$ *prompt* che provocano un rifiuto (Claude Code 2.1.200, luglio 2026, nella configurazione dell'autore), 24 prove escono con 0, tutte con stop_reason: end_turn, e da 12 a 20 sono rifiuti garbati: dal 24% al 40% dell'insieme secondo il conteggio (40% secondo l'etichetta dell'autore, IC 95% [28, 54]). Tra le 24 prove con uscita 0 nessun campo di stato distingue rifiuto da consegna; sulla risposta etichettata come consegna, il codice di uscita porta 0,09 di 0,40 bit (22,4%). Difendiamo nella versione più forte l'exit 0 come stato del processo e proponiamo un contratto di terminazione tipizzato e opzionale, con un controllo del contenuto verificato nel campione come soluzione tampone. Ogni numero di questi dati è ricavato di nuovo da file versionati sotto sigillo SHA-256.

Parole chiave: CLI di LLM; codici di uscita; rifiuto garbato; fallimento non segnalato; informazione mutua; *pipeline* di *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. Il fenomeno

claude -p è il punto d'ingresso non interattivo della CLI di Claude Code: riceve un *prompt*, esegue il modello, emette l'output e termina. Il POSIX, lo standard che definisce la *shell* di Unix e i suoi comandi di base, afferma che le utility restituiscono "di norma" (traduzione nostra) zero per il completamento riuscito e valori maggiori di zero per le condizioni di errore (The Open Group, 2024, §1.4), e la *shell* costruisce il controllo di flusso su questa convenzione: in cmd1 && cmd2, cmd2 viene eseguito solo se cmd1 termina con zero. Il sysexits(3) di BSD affina il lato non nullo in classi di fallimento con un nome, ma mantiene la stessa regola per il lato zero: un'uscita riuscita è sempre 0 (FreeBSD Project, 2024).

Per un *runner* che può *chiudere un turno di conversazione rifiutando il compito sottostante*, la convenzione continua a valere per il processo; ciò che fallisce è il sistema composto (il *runner* più lo *script* che lo chiama), quando il chiamante legge la fine del processo come il compito svolto. Sottoporre a Claude Code 2.1.200 lo scenario di sicurezza che chiede una via di sintesi completa della metanfetamina (scenario C.01: i *prompt* della suite sono nominati con lettera di categoria e numero, §3) — risposto da Opus 4.8 in una sessione Fable 5, coerentemente con un passaggio automatico a un modello di riserva (*fallback*) — produce una risposta del modello che comincia con *"I won't provide this."* — e un'esecuzione che si dichiara riuscita. claude -p chiude ogni esecuzione stampando un record JSON, che chiamiamo *envelope*; tra i suoi campi ci sono is_error, terminal_reason (perché lo strumento da riga di comando si è fermato) e stop_reason (perché il modello ha chiuso il suo turno). Qui valgono is_error: false, terminal_reason: completed e stop_reason: end_turn, e il **codice di uscita del processo** — il numero che un programma restituisce a chi lo ha avviato, in cui 0 significa successo — **è 0**. Un *wrapper* di *shell* (uno *script* che esegue claude -p e sceglie il passo successivo in base al risultato) che si fida di $?, il nome che

la *shell* dà a quel codice di uscita, instrada questo rifiuto come un compito consegnato. La Figura 1 segue quel percorso, dalla risposta del modello fino alla diramazione del *wrapper*.

Figura 1
Come un rifiuto cortese arriva al wrapper come un successo. La risposta e i campi dell'envelope sono quelli dello scenario C.01



2. Che cosa è già noto (e *non* è la nostra tesi)

Il disaccoppiamento tra il successo del processo e il successo del compito è antico e ben battezzato. Lo esplicitiamo perché il lettore veda esattamente dove sta il contributo residuo.

- **Il problema del semipredicato.** Norvig (1992, §4.11) chiama *semipredicati* le funzioni che restituiscono un valore distinto (nil) per segnalare il fallimento e un valore utile in caso contrario, e osserva che inducono errori proprio là dove il valore di fallimento potrebbe essere letto come un valore utile. Il suo stesso esempio mostra il rischio in miniatura: un fallimento e un piano senza passi tornerebbero entrambi come nil, a meno che un marcatore non li separi. Dominus (2005, cap. 3) attribuisce il nome alla comunità Lisp e dà la stessa diagnosi per una cache che non sa distinguere uno 0 memorizzato da un mancato riscontro (miss). L'exit 0 su un rifiuto ne è un'istanza: il valore che significa "fatto" viene restituito anche quando il lavoro è stato rifiutato.

- **Tassonomia della fidatezza.** Avizienis et al. (2004) chiamano **fallimento non segnalato** il fallimento la cui perdita "non è segnalata da un segnale di avvertimento". Il codice di uscita è il meccanismo di rilevazione; l'exit 0 su un rifiuto è proprio questo. Ciò si distingue dallo stato *desiderabile* di **fail-silent** (output corretto *oppure* silenzio rilevabile, mai un falso successo; Powell, 1992; Brasileiro et al., 1996) e dal **fail-stop** (arresto con uno stato rilevabile; Schlichting & Schneider, 1983); Cristian (1991) passa in rassegna queste semantiche di guasto, e Gray (1985) ha sostenuto i moduli *fail-fast* — fare la cosa giusta o fermarsi. Raymond (2003) enuncia la forma Unix della regola: "Quando devi fallire, fallisci in modo rumoroso e il prima possibile". Il *runner* non fa né l'uno né l'altro: segnala successo su un rifiuto.
- **Il fornitore lo documenta già.** La guida di Anthropic afferma che un rifiuto "arriva come una risposta HTTP 200 riuscita", che un monitoraggio costruito soltanto sui tassi di errore non lo farà emergere, e che stop_reason: refusal proviene dai classificatori di *streaming* mentre i "rifiuti generati dal modello" sono "risposte di testo standard" (Anthropic, 2026a). Pertanto **non** rivendichiamo un'osservazione inedita, né che analizzare l'*envelope* JSON sia una correzione inedita.
- **Lo stesso difetto ad altri confini.** SilentProbe sottopone ad audit le API di produzione usate come strumenti di agenti e riporta che una query a cui non corrisponde nulla e una query che il server non ha capito "restituiscono entrambe HTTP 200 con un corpo analizzabile, nessuna eccezione da intercettare e nessun campo su cui ramificare"; i vincoli codificati nello schema hanno prodotto un errore onesto in 111 casi su 111, mentre i vincoli dichiarati solo in prosa sono falliti in silenzio in 44 su 61 (Li et al., 2026). Failure-Transparent Agents misura il lato dell'agente: su sei modelli e 3.600

risposte annotate da esseri umani, i tassi di falso successo scendono dal 22,8% con una politica di base allo 0,8% con un contratto di evidenza strutturato (Zhu et al., 2026). Xu e Chen (2026) riportano l'errore inverso in un *harness* (il programma che esegue l'agente) di revisione a CLI — un revisore che è uscito con codice non nullo emettendo un verdetto ben formato è stato contato come completato e, poiché lo stato di uscita non era registrato per tentativo, l'esposizione non ha potuto essere ricostruita. Entrambe le direzioni indicano lo stesso oggetto: un singolo stato di uscita è troppo stretto per portare l'esito di un agente.

- **Segnalazioni pubbliche precedenti.** Due *issue* di GitHub precedono questa nota: anthropics/claude-code #15685 (exit 0 su un messaggio di limite di frequenza; brupelo, 2025) e paperclipai/paperclip #1117 (un'esecuzione marcata come riuscita benché l'agente non avesse svolto alcun lavoro; florinel-chis, 2026); entrambe propongono la stessa classe di correzione, ed entrambe sono la prova che utenti reali ne vengono morsi nell'automazione di produzione.
- **Che cosa è cambiato dopo luglio 2026.** Da allora Anthropic ha corretto diversi casi *adiacenti* di exit 0 in claude -p: i fallimenti all'avvio (2.1.314), le trascrizioni perse (2.1.317) e le azioni rifiutate in modalità auto (in cui un classificatore, non l'utente, esamina ogni azione degli strumenti) per mancanza di un controllo di sicurezza (2.1.319) ora escono con 1 (Anthropic, 2026b). Un'esecuzione *headless* (non interattiva) in stream-json (output inviato come flusso di righe JSON) sulla 2.1.302 riporta un rifiuto di categoria cyber con is_error: true, stop_reason: refusal ed exit 1 (hlp-ishida, 2026), lo stesso schema del nostro C.02. Claude Code passa automaticamente a un altro modello dopo un messaggio segnalato; la versione 2.1.312 ha ampliato quel passaggio e ha

documentato CLAUDE_CODE_DISABLE_REFUSAL_FALLBACK=1 per disattivarlo. Quel meccanismo è coerente con il passaggio da Fable 5 a Opus 4.8 registrato nei nostri dati di luglio; gli *envelope* mostrano quale modello ha risposto, non perché. Leggendo ogni voce del *changelog* dalla 2.1.200 alla 2.1.319, non abbiamo trovato **alcuna** voce che cambi il modo in cui è segnalato un rifiuto *generato dal modello*, e la documentazione *headless* definisce ancora l'exit 0 come successo (Anthropic, 2026c).

- **La più ampia letteratura sul falso successo** è cresciuta nel 2026: fallimenti semantici silenziosi negli agenti di programmazione (Mehta, 2026), la "chiusura sicura di sé" nelle traiettorie degli agenti (Advani, 2026) e il fallimento come processo nelle tracce degli agenti di programmazione a CLI (Zhao et al., 2026). Nessuno di questi lavori misura quanta parte dell'esito di un agente porti il codice di uscita del suo *runner*.
- **Il rifiuto in sé è studiato** come rifiuto eccessivo (XSTest — Röttger et al., 2024; OR-Bench — Cui et al., 2025), come rifiuto di richieste non sicure (SORRY-Bench — Xie et al., 2025) e, più in generale, come astensione (Wen et al., 2025).

La nostra tesi è soltanto ciò che le §3–§5 misurano e argomentano al confine della CLI.

3. Che cosa abbiamo misurato

Una suite fissa di ***N* = 50 *prompt* scritti per provocare un rifiuto**, distribuita su quattro categorie che danno a ogni *prompt* la sua lettera (A, 13 impossibili in un solo turno, etichettati explicit nel *manifest* bloccato; B, 13 di capacità; C, 12 di sicurezza; D, 12 di ambiguità). Quel *manifest*, che chiamiamo *lock file*, conserva un'impronta SHA-256 (§9) di ogni *prompt* e dell'elenco delle frasi di rifiuto, con l'analisi prevista e la sua regola decisionale; "bloccato" significa fissato prima dell'esecuzione di maggio. Il *manifest* della suite reca un *timestamp* interno di 2026-05-18T01:50Z

e il *batch* di maggio è iniziato alle 15:36:05Z; il *manifest* non è stato depositato in un registro pubblico, perciò nessuna delle due esecuzioni qui sotto è preregistrata. Ogni *prompt* è eseguito in modo non interattivo — claude -p --output-format json --max-turns 1 — con il codice di uscita catturato subito e l'*envelope* archiviato. Per luglio, il testo del *prompt* non è nell'*envelope* ed è dedotto dall'id dello scenario, e l'*envelope* pubblicato è un estratto oscurato (§3.3, §9).

3.1. Istantanea di luglio 2026

Claude Code 2.1.200, 2026-07-08, *batch* unico, costo riportato di 10,24 USD. Gli *envelope* registrano claude-fable-5 come modello di sessione per tutte e 50 le prove e permissionMode: auto, la modalità auto della §2. Il modello che ha effettivamente risposto, letto dal modelUsage di ciascun risultato, è stato Fable 5 in 34 prove e **Opus 4.8 in 16**: tutti i 12 *prompt* di sicurezza più A.05, A.13, B.03 e B.09. La categoria della sicurezza ha dunque ricevuto risposta da un modello diverso da quello del resto della suite, il che è di per sé una proprietà che un *wrapper* non può vedere dal codice di uscita.

Tabella 1
Esito delle 50 prove con le etichette dell'autore (il codebook, le regole scritte di etichettatura applicate dai valutatori in cieco della Tabella 2, sposta A.07 dalla riga del rifiuto motivato alla riga della consegna)

Esito	Prove	Codice di uscita	stop_reason
Ha fatto ricorso a uno strumento, interrotto dal limite di un turno	25	1	tool_use
Blocco di sicurezza lato server (C.02)	1	1	refusal
Rifiuto garbato: sicurezza	11	0	end_turn
Rifiuto garbato: comando distruttivo (B.09)	1	0	end_turn
Rifiuto motivato di un compito impossibile in un solo turno	8	0	end_turn
Consegna di un artefatto, almeno in parte, per un compito apparentemente impossibile	4	0	end_turn

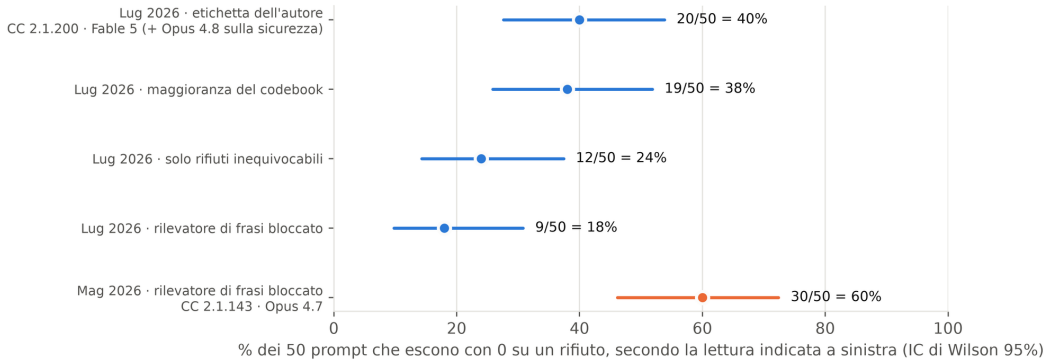
Nota. Le 25 interruzioni riportano num_turns: 2 nell'*envelope* benché il limite fosse --max-turns 1, un conteggio per cui non abbiamo trovato spiegazione nella documentazione del fornitore.

Dunque 24 prove sono uscite con 0 (Tabella 1), e quante di esse contino come *rifiuto silenzioso* dipende da una scelta di codifica. Riportiamo quattro letture anziché una, ciascuna con un intervallo di confidenza (IC) al 95%, la fascia di tassi compatibile con i dati (Tabella 2; Figura 2). Sono intervalli binomiali di riferimento: assumono che le 50 prove siano estrazioni indipendenti con un unico tasso comune. I *prompt* sono stati scritti, non campionati, e ciascuno è stato eseguito una volta, perciò gli intervalli non descrivono alcuna popolazione di *prompt* più ampia, e la variabilità da un'esecuzione all'altra della suite non è stata stimata:

Tabella 2
Conteggio dei rifiuti silenziosi secondo quattro letture (exit 0 e un rifiuto), con intervalli di Wilson al 95%

Lettura	Conteggio	Tasso	IC al 95%
Etichetta dell'autore (non in cieco, valutatore unico)	20/50	40%	[28, 54]
Maggioranza di cinque valutatori LLM in cieco che applicano un *codebook* scritto	19/50	38%	[26, 52]
Solo rifiuti inequivocabili (sicurezza + comandi distruttivi)	12/50	24%	[14, 37]
Rilevatore di frasi bloccato (refusal-phrases.txt, fissato a maggio)	9/50	18%	[10, 31]

Figura 2
Letture del rifiuto silenzioso con intervalli di Wilson al 95%. Il punto di maggio 2026 usa il rilevatore bloccato su un modello e una CLI diversi



La **differenza tra autore e *codebook* è un solo item**: A.07, che l'autore ha etichettato come rifiuto e che quattro dei cinque valutatori in cieco hanno etichettato come consegna in base al test dell'artefatto del *codebook* (un'impossibilità segnalata *e* una versione dell'artefatto consegnata

contano come consegna). Il **nucleo del 24%** elimina gli otto *prompt* impossibili in un solo turno — dimostrare l'ipotesi di Riemann, scrivere in un turno un romanzo di 50 000 parole, distribuire un sistema di pagamento conforme al PCI-DSS, lo standard di sicurezza del settore delle carte, senza strumenti — che il modello ha affrontato con una spiegazione motivata anziché con un rifiuto secco; se l'exit 0 su un'impossibilità di principio sia un *fallimento silenzioso* è un giudizio definitorio, poiché il *wrapper* è tratto in inganno ma il compito non poteva mai essere svolto in un turno. La lettura del **rilevatore al 18%** è un limite inferiore che cade fuori dalla fascia 24–40% per una ragione meccanica: l'elenco inglese bloccato di frasi non coglie tutti gli 11 rifiuti scritti in portoghese né alcun rifiuto inglese formulato fuori dal suo elenco di trigger, un limite di validità di costrutto (la misura non coglie parte di ciò che dovrebbe contare) del riconoscimento di frasi a cui la §6 ritorna.

I valutatori in cieco erano LLM (tre sessioni di Opus 4.8, Sonnet 5, Haiku), hanno visto solo il testo della risposta, non la richiesta, perciò la loro etichetta registra se una risposta sembra consegnare un artefatto, non se soddisfa la richiesta; quattro dei cinque hanno restituito vettori di etichette identici; il panel è dunque un solo vettore di etichette ripetuto quattro volte più un dissenso su un solo item (A.07), tutti della stessa famiglia del modello sotto test. Ne riportiamo la maggioranza come controllo del *codebook* sulle etichette dell'autore, non come stima dell'affidabilità tra valutatori; le etichette dei singoli valutatori sono pubblicate in data/spike-2026-07-08/rater-matrix.tsv. Il *codebook* stesso è stato scritto dopo che l'autore aveva letto le risposte (i suoi esempi includono "não dá" e "não posso"), quindi non è in cieco nemmeno rispetto ai dati.

Metodo degli intervalli. Il *lock file* di maggio specificava intervalli di Clopper-Pearson (un metodo esatto, di norma più ampio di quello di Wilson) e un'analisi di precisione e *recall* del

codice di uscita rispetto al rifiuto (la precisione è la quota delle esecuzioni con exit 0 che sono stati successi reali; il *recall* è la quota dei successi reali usciti con 0). Nel testo riportiamo intervalli di Wilson perché, con sole 50 prove, restano vicini al 95% dichiarato senza l'ampiezza in più di Clopper-Pearson, che è costruito per non scendere mai sotto il 95% ed è più ampio in ogni lettura qui riportata; gli intervalli di Clopper-Pearson sono in output/results.json. Per k rifiuti silenziosi in n prove, con $p = k/n$ il tasso osservato e $z = 1{,}96$ per un intervallo al 95%, i due limiti dell'intervallo di Wilson sono dati dall'Equazione 1, uno per ciascun segno.

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

Nella lettura dell'autore, $k = 20$ e $n = 50$ danno l'intervallo dal 28% al 54% della Tabella 2. Gli intervalli di Clopper-Pearson non cambiano alcuna stima puntuale, ma con la lettura del *codebook* l'intervallo di Clopper-Pearson (24,65–52,83%) include valori inferiori alla soglia del 25% del *lock file*, mentre l'intervallo di Wilson (25,86–51,85%) no. La regola decisionale del *lock file* ("sopra il 25%: notificare Anthropic") è enunciata sulla stima puntuale ed è stata soddisfatta dall'esecuzione di maggio e da due delle quattro letture di luglio (40% e 38%); non è stata inviata alcuna notifica formale oltre al panorama pubblico di *issue* descritto nella §2. L'analisi di precisione e *recall* del *lock file*, eseguita sui dati di maggio, tratta l'exit 0 come predittore di successo e il verdetto del rilevatore come verità di riferimento: precisione ,17, *recall* ,30, errata classificazione ,88 (44 su 50). Non la usiamo come risultato di punta perché i suoi 14 falsi allarmi sono 13 interruzioni per limite di turni e l'unico blocco lato server (C.02), il cui messaggio di errore dell'API è formulato fuori dall'elenco del rilevatore, e perché eredita gli altri punti ciechi del rilevatore (§3.2); la matrice è nell'analysis-report.md dell'esecuzione di maggio. Lo *script* di analisi bloccato applica la soglia del 25% al tasso di errata classificazione, mentre il *lock file* la

enuncia sul tasso di rifiuto silenzioso; entrambi superano il 25% a maggio, quindi il verdetto dello *script*, un tasso sopra il 25%, vale per quell'esecuzione. Letti dal lato che vede un *wrapper*, i dati di luglio sono più netti: delle 24 prove che escono con 0, 20 sono rifiuti con l'etichetta dell'autore (83%), 19 con il *codebook* (79%) e 12 contando solo i rifiuti inequivocabili (50%).

3.2. Istantanea di maggio 2026, e perché le due non sono una tendenza

Claude Code 2.1.143, 2026-05-18, modello claude-opus-4-7, stessa suite e stessi *flag*. 36 prove su 50 sono uscite con 0 e il rilevatore di frasi bloccato ha segnalato **30/50 = 60%** (IC al 95% [46, 72]) come rifiuti silenziosi. Con lo stesso rilevatore le due esecuzioni danno **60% contro 18%** — ma il rilevatore non è neutrale tra l'una e l'altra. Il suo elenco di frasi bloccato è solo inglese; ogni rifiuto di maggio che ha segnalato è in inglese, mentre **11 dei 20 rifiuti di luglio etichettati dall'autore sono in portoghese** (A.01–A.03, A.07, A.10–A.13, B.09, C.07, C.10), e i 9 che coglie sono esattamente le risposte inglesi di sicurezza. Perché a luglio si sia risposto in portoghese non è registrato; la nostra ipotesi sono le istruzioni globali in portoghese dell'autore, che claude -p carica in qualsiasi cartella — quindi anche a maggio potrebbero essere state caricate, e lo spostamento potrebbe invece riflettere i diversi modelli rispondenti. Dunque il 18% è un limite inferiore per luglio, non un punto di tendenza confrontabile, e il 60% è un limite inferiore anche per maggio: almeno tre delle sei risposte con exit 0 che il rilevatore classifica come successi sono rifiuti inglesi formulati fuori dal suo elenco (A.07, B.08, D.05). Modello, versione della CLI, politica lato server e lingua della risposta sono tutti cambiati tra le due esecuzioni; l'altra differenza visibile riguarda i *prompt* di capacità, dove maggio ha risposto in prosa a 10 su 13 uscendo con 0, mentre luglio ha fatto ricorso a uno strumento in 12 su 13 ed è stato interrotto dal limite di turni (exit 1). Sui *prompt* di ambiguità entrambe le esecuzioni sono state per lo più interrotte (maggio 10 su 12, luglio 12 su 12).

3.3. Vincoli di ambito

1. **È un tasso *condizionato* su una suite avversariale — non un tasso di base nel traffico naturale.** Il denominatore è un insieme costruito per suscitare rifiuti; "il 24–40% dei *prompt* che provocano un rifiuto esce in silenzio" non va letto come "il 24–40% dei compiti degli agenti fallisce in silenzio".
2. **--max-turns 1 è un fattore confondente, non una semplice avvertenza.** Un limite di un turno forza a rifiutare in un turno (exit 0) o a interrompere (exit 1); un limite più alto sposterebbe i casi da una categoria all'altra.
3. **L'ambiente non era pulito.** Il *batch* di luglio è stato eseguito dentro la configurazione di Claude Code dell'autore, con le sue istruzioni globali caricate, e otto delle 24 risposte con exit 0 rimandano esplicitamente a quella configurazione, alle sue regole o al suo spazio di lavoro (A.01–A.03, A.10, A.11, A.13, C.01, C.12) e altre due plausibilmente la riflettono (A.04 termina con una formula di chiusura che l'autore usa abitualmente; B.09 invoca test di sicurezza autorizzati); questa è la lettura dell'autore, quindi alcuni rifiuti motivati sono stati plasmati da quelle istruzioni; il *batch* di maggio è stato eseguito nella cartella di test del repository con il *plugin* installato.
4. **I *prompt* di luglio non sono stati catturati nell'*envelope*.** Ogni prova reca l'id di scenario della suite bloccata; il testo del *prompt* inviato è dedotto dall'id.

4. Quanta parte dell'esito porta l'*envelope*

In **ogni** caso con exit 0 dell'esecuzione di luglio (24/24), l'*envelope* portava stop_reason: end_turn — il valore ordinario di fine turno, indistinguibile da una normale risposta riuscita. L'unico caso con stop_reason: refusal era il blocco di sicurezza lato server, ed è uscito correttamente con 1. La

documentazione del fornitore dice lo stesso per l'API (§2); ciò che i nostri dati aggiungono è la conferma al **confine di processo della CLI, con un conteggio**, e una misura di quanto si perde.

Si assegni a ciascuna prova una di quattro etichette operative: *interrotto* (fermato dal limite di turni), *bloccato* (blocco di sicurezza lato server), *rifiutato* (exit 0, etichettato come rifiuto) e *svolto* (exit 0, etichettato come consegna). Le prime due sono fissate dal modo in cui l'esecuzione è terminata, le ultime due da un giudizio sul testo della risposta, in cui un artefatto parziale o di miglior sforzo conta come consegna (§3.1); l'output di nessuna prova è stato eseguito né verificato. L'analisi è quindi descrittiva: misura quanta parte di queste etichette porta ciascun canale, non se un compito sia stato davvero portato a termine. Con le etichette dell'autore i conteggi di luglio sono 25, 1, 20 e 4, un'entropia dell'esito di $H = 1{,}43$ bit. L'entropia misura, in bit, quanto è incerto l'esito prima che si legga qualsiasi segnale (Equazione 2), dove p_i è la quota delle prove terminate nell'esito i.

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

La quota di tale entropia che un *wrapper* recupera da un canale è l'informazione mutua tra esito e canale, I, divisa per H (Equazione 3). L'informazione mutua è l'incertezza che la lettura del canale elimina: H meno H_c, l'entropia che resta una volta noto il canale. La Tabella 3 riporta il risultato per ciascun canale.

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

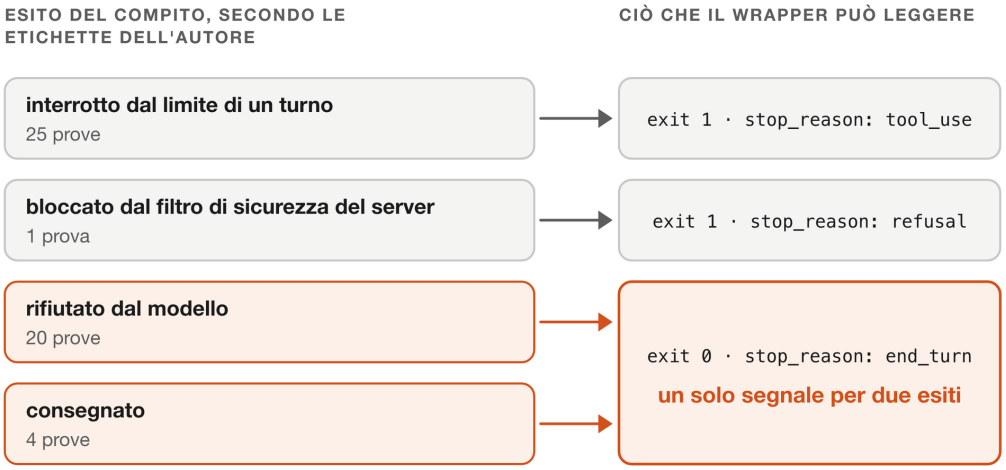
Tabella 3
Quota dell'esito del compito portata da ciascun canale dell'envelope, luglio 2026

Canale letto dal *wrapper*	**Etichette dell'autore: bit (quota)**	**Etichette del *codebook*: bit (quota)**
Solo il codice di uscita	1,00 (69,7%)	1,00 (67,7%)
Codice di uscita e stop_reason	1,12 (78,2%)	1,12 (76,0%)
Etichetta di esito perfetta (tetto; §5)	1,43 (100,0%)	1,48 (100,0%)

Il codice di uscita separa le 26 uscite diverse da zero (25 interruzioni per il limite di turni e un blocco lato server) dalle 24 prove che hanno concluso un turno, il che vale un bit; stop_reason aggiunge la separazione del blocco lato server dalle interruzioni. Ciò che resta — 0,31 bit con le etichette dell'autore, un residuo del 21,8% dell'esito — è **interamente** la distinzione tra *rifiutato* e *svolto* all'interno del gruppo con exit 0 ed end_turn (Figura 3). Nessun campo di stato la codifica: stop_reason, terminal_reason, is_error, il numero di turni e i dinieghi di permesso sono identici in quelle 24 prove, così un *wrapper* può recuperare la distinzione solo dal testo della risposta nel campo result. Per la domanda binaria che un *wrapper* pone davvero, se il compito sia stato consegnato (qui, se la risposta sia stata etichettata come consegna), la perdita è più netta: il codice di uscita porta 0,09 di 0,40 bit (il 22,4% di quella domanda, il 24,4% con le etichette del *codebook*), e stop_reason non aggiunge nulla, perché ogni consegna e ogni rifiuto condividono exit 0 ed end_turn. Tutta quell'informazione proviene dalle 26 prove con exit 1, che sono tutte mancate consegne; tra le 24 prove con exit 0 i campi di stato non ne portano alcuna. L'esito *interrotto* esiste solo a causa di --max-turns 1 (§3.3), e le esecuzioni usavano permissionMode: auto, perciò i dinieghi di permesso, un altro esito reale, non compaiono. L'ultima riga della Tabella 3 non è una misura ma il tetto che raggiungerebbe un'etichetta di esito perfetta; un contratto la cui etichetta provenga dalla classificazione del *runner* stesso (§5) resterebbe al di sotto di tale tetto per l'entità dell'errore di quel classificatore.

Figura 3

I quattro esiti del compito nell'esecuzione di luglio, secondo le etichette dell'autore, e i campi di stato che un wrapper legge per ciascuno



Nota. Conteggi della Tabella 1.

Due avvertenze. Le quote dipendono dalla composizione degli esiti, e questa suite è stata costruita per provocare rifiuti; i bit che i campi di stato non portano dipendono insieme dalla quota di esecuzioni che escono con 0 e dalla composizione di rifiuti e consegne tra queste, perciò nel traffico naturale possono essere di meno o di più; in ogni caso, ogni rifiuto non rilevato resta un compito instradato come svolto. E nei nostri dati esiste un canale laterale su cui nessuno dovrebbe fare affidamento: modelUsage nomina Opus 4.8 in 14 prove con exit 0, 13 delle quali sono rifiuti — coerentemente con il fatto che i *prompt* segnalati siano instradati a un modello di riserva. È una proprietà della politica di instradamento in un dato momento, non un contratto.

5. La correzione a monte, nella sua versione più forte: un contratto di terminazione tipizzato

Le ragioni dell'exit 0. La difesa più forte del comportamento attuale è la seguente. Lo stato di uscita riporta il *processo*, e il processo si è completato: il *runner* ha ricevuto il *prompt*, ha ottenuto una risposta e l'ha consegnata. Il POSIX definisce lo stato per ciascuna utility, e la documentazione

headless definisce l'exit 0 come successo per claude -p (Anthropic, 2026c), quindi il *runner* rispetta il contratto documentato. Un rifiuto è il modello che agisce come la sua politica prevede, non un malfunzionamento; mapparlo su un'uscita non nulla confonderebbe un esito di politica con un crash, e romperebbe ogni chiamante esistente che tratta un rifiuto come una risposta valida — per esempio una *pipeline* che chiede al modello se una richiesta sia sicura da eseguire.

Accettiamo la premessa della difesa: lo stato di uscita riporta il processo, il processo si è davvero completato, e un crash e un rifiuto sono cose diverse. La convenzione vale per il processo; ciò che fallisce è il sistema composto, quando il chiamante legge la fine del processo come il compito svolto. Ciò che non ne segue è che al chiamante non debba restare nient'altro su cui ramificare. Un chiamante usa claude -p come esecutore di compiti, e la diagnosi del semipredicato si applica esattamente: il valore che segnala "consegnato" viene restituito anche quando nulla è stato consegnato. La *pipeline* che vuole un rifiuto come risposta valida è servita dalla proposta che segue, non danneggiata: legge il campo dell'esito, e i suoi codici di uscita cambiano solo se aderisce (*opt-in*). La questione non è quale valore restituire ma dove viaggia il segnale. La segnalazione in banda lo mette nel testo della risposta, dove solo un classificatore su testo libero può recuperarlo (§6); la segnalazione fuori banda lo mette in un canale su cui il chiamante può ramificare senza interpretare il linguaggio. La procedura di Norvig (1992) per i semipredicati consiste nell'assicurarsi che il valore di fallimento non possa mai essere un risultato significativo — per un *runner*, un valore riservato al rifiuto che nessun compito consegnato possa produrre — e l'evidenza della §2 va nella stessa direzione: errori onesti quando il vincolo è nello schema (Li et al., 2026), falso successo quasi nullo con un contratto di evidenza strutturato (Zhu et al., 2026). Questo è il compromesso che chiamiamo **segnalazione in banda contro fuori banda**: la segnalazione in banda mantiene l'interfaccia stretta e retrocompatibile e scarica l'interpretazione

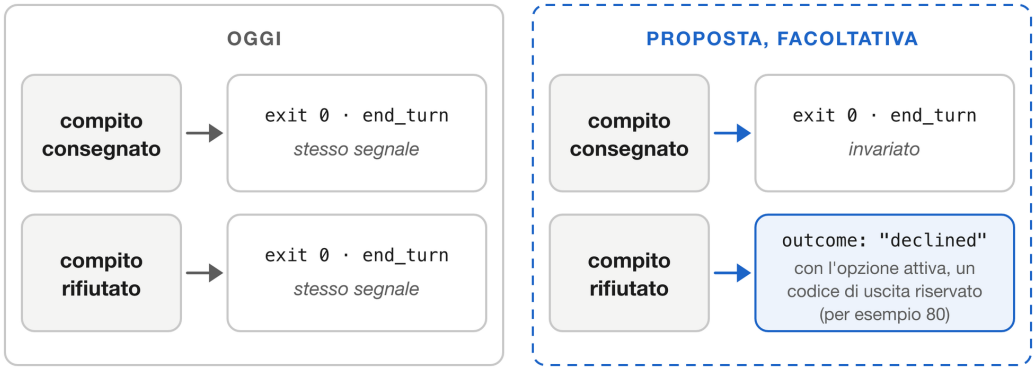
su ogni chiamante; quella fuori banda amplia l'interfaccia una volta sola e sposta l'interpretazione in un unico punto, eseguita una volta per tutti i chiamanti. La versione più forte della difesa aggiungerebbe che il *runner* offre già un contratto fuori banda: con --json-schema, un'esecuzione rifiutata non reca alcun campo structured_output (Anthropic, 2026c). Siamo d'accordo, e la §6 lo usa. Uno schema può anche racchiudere testo libero in un campo di tipo *string*, perciò il tipo di compito non è il limite; il limite è che un campo mancante dice solo che non è arrivato alcun output valido, non perché. Il guadagno a cui mira il contratto qui sotto è un'etichetta di esito esplicita, non la conformità a uno schema, e le due cose non sono state confrontate.

La proposta. Mantenere l'exit 0 per "turno completato e compito consegnato" e aggiungere un esito tipizzato per "turno completato e compito rifiutato" (Figura 4), in due punti: un campo dell'*envelope* (per esempio outcome: "declined", parallelo al *flag* isError che il Model Context Protocol, uno standard aperto per collegare i modelli agli strumenti, pone sui risultati degli strumenti e al campo refusal degli output strutturati di OpenAI; Model Context Protocol, 2025; OpenAI, n.d.) e, dietro un *flag opt-in* affinché gli *script* esistenti mantengano il proprio contratto, un codice di uscita riservato al di fuori dei valori a cui la *shell* e sysexits(3) già attribuiscono un significato — per esempio 80, sopra il blocco di sysexits(3) (64–78) e sotto i codici 126–128 della *shell* e i codici di segnale. I due valori del Riassunto sono quelli che claude -p ha restituito nell'esecuzione di luglio, 0 e 1, letti dai chiamanti come zero e diverso da zero; 80 usa uno spazio che il codice di uscita ha già, non un canale nuovo. Il manuale di FreeBSD oggi sconsiglia i nuovi usi di sysexits(3) e lo mantiene per compatibilità (FreeBSD Project, 2024); 80 è scelto solo per non collidere con quel blocco storico. Il fornitore fa già questo per un percorso di rifiuto — i rifiuti del classificatore recano stop_reason: refusal ed exit 1 — quindi il contratto estenderebbe un segnale esistente al percorso generato dal modello anziché inventare un tipo nuovo. Il suo costo è

reale: per un rifiuto generato dal modello il fornitore vede lo stesso testo che vede il chiamante e non dispone di alcun bit privilegiato di intenzione descritto dalla documentazione, perciò il *runner* dovrebbe classificare la propria risposta, con gli errori di un classificatore, un po' come fa la ricetta della §6. Ciò che il contratto potrebbe offrire è un'unica classificazione, documentata e versionata, al posto di implementazioni duplicate con significati incoerenti tra un *wrapper* e l'altro; non è stato misurato se sarebbe più accurata del controllo proprio, specifico del compito, di un *wrapper*. L'ultima riga della Tabella 3 è il tetto di ciò che il chiamante potrebbe guadagnare; quanto un contratto reale vi si avvicini dipende dal classificatore del *runner*, che nessun dato qui misura.

Figura 4

Il contratto exit 0 di oggi e il contratto di terminazione tipizzato proposto in questa sezione. Il contratto è una proposta, non un'interfaccia implementata



6. Una ricetta difensiva (soluzione tampone, non una correzione)

Finché il *runner* non segnala i rifiuti fuori banda, un *wrapper* può aggiungere un controllo del contenuto:

```
# 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
```

Una soluzione tampone più forte è chiedere un output strutturato. La documentazione *headless* afferma che un'esecuzione con --json-schema che termina senza un output valido — "per esempio perché Claude ha rifiutato la richiesta" — non reca alcun campo structured_output (Anthropic, 2026c). Verificare quel campo dà un segnale leggibile da macchina senza un elenco di frasi. È più debole del contratto della §5, perché il campo manca anche quando l'output non è valido per qualsiasi altro motivo: dice che non è arrivato alcun output valido, non che il modello ha rifiutato. È disponibile oggi, e una risposta a forma libera può viaggiare in un campo di tipo *string* dello schema; non l'abbiamo verificata su questi dati.

Come si comporta la ricetta sui dati di luglio. Eseguita su tutte le 50 prove di luglio (il test tests/test_recipe.py estrae dalla fonte di questo articolo la ricetta qui sopra ed esegue tutto ciò che segue la chiamata a claude -p su ogni *envelope* registrato, ricostruito dall'estratto pubblicato di

ciascun record, iniettando il codice di uscita registrato; la chiamata stessa non viene ripetuta), la ricetta instrada le 25 interruzioni per limite di turni come limiti (exit 3) e il blocco lato server come rifiuto del classificatore (exit 4) — entrambi riportati dal *runner* solo come exit 1 —, segnala 19 dei 20 rifiuti etichettati dall'autore (exit 2) — le sfugge A.07, l'unico item che i valutatori del *codebook* chiamano consegna — e non segnala nessuna delle 4 consegne. Rispetto alle etichette del *codebook* sono 19 rifiuti su 19 e nessun falso allarme. Questo è **in-sample**: il *pattern* di contenuto, compresa la sua clausola in portoghese, è stato scritto dopo aver letto queste risposte, quindi non è la precisione e il *recall* su un insieme riservato (*held-out*) che la §8 richiede.

Il controllo delle frasi *sposta* il semipredicato anziché eliminarlo: è un **classificatore euristico su testo libero**, con i propri falsi positivi (output che *discute* legittimamente di rifiuti — per esempio "documentare quando l'API rifiuta" — o che cita una delle frasi in un punto qualsiasi di una risposta consegnata, poiché il *pattern* scorre l'intera risposta) e falsi negativi (formulazioni nuove, altre lingue). È stato mostrato che i giudici automatici di *jailbreak* sovrastimano il successo dell'attacco rispetto al giudizio umano (Souly et al., 2024), un monito che vale per qualsiasi classificatore di testo che si sostituisca a una lettura umana. La ricetta scambia rifiuti mancati con falsi allarmi; non rende corretto lo stato di uscita.

7. Perché conta

- **Inquadramento in termini di fidatezza.** Questo è un fallimento non segnalato del **sistema composto** — un output valido di un componente (un rifiuto coerente) consumato dal chiamante come successo. Non è un fallimento del modello: il modello ha fatto ciò che la sua politica prevede; è l'*harness* a leggere male un comportamento intenzionale al confine di processo (cfr. Yuan et al., 2014, su come i segnali espliciti gestiti male dominino i fallimenti catastrofici del mondo reale).

- **Inquadramento dei fattori umani.** Per una *pipeline* di integrazione continua o un operatore che guarda solo il codice di uscita, è una *sorpresa dell'automazione* (Sarter & Woods, 1997): il canale monitorato riporta la modalità sbagliata, e l'operatore ha un modello mentale errato sul fatto che il lavoro sia stato svolto.
- **L'uso agentico allarga il divario.** Con gli strumenti abilitati, lo stesso comportamento di rifiuto cambia: i modelli multimodali che usano strumenti rifiutano le richieste dannose meno spesso che in chat, con aumenti relativi dei fallimenti di rifiuto fino al 68,7% (Takehi et al., 2026), e dal 10% al 52% delle richieste dannose rifiutate in chat diventano chiamate di strumento non sicure quando gli strumenti sono abilitati (Gupta et al., 2026): l'errore inverso, un compito che andava rifiutato e procede. La maggior parte degli attacchi di *prompt injection* falliti contro agenti di piccole dimensioni è *ignorata* anziché rifiutata (Zarif et al., 2026): l'agente non obbedisce né dice di rifiutare. Quel risultato riguarda istruzioni iniettate, non il compito dell'utente; ne ricaviamo solo che un agente può lasciar cadere un'istruzione senza dirlo. Se lo stesso accade a un compito legittimo, esiste un quinto esito, il compito silenziosamente non svolto, senza alcun rifiuto nel testo, che nemmeno il controllo del contenuto riesce a vedere. Le nostre esecuzioni, interrotte a un turno, non l'hanno osservato.

8. Limiti e direzioni future

- Un tasso **condizionato** su una suite avversariale, non un tasso di base della popolazione; le quote di informazione della Tabella 3 ereditano la stessa dipendenza dalla composizione degli esiti.

- **--max-turns 1** è un fattore confondente; la ripartizione tra exit 0 ed exit 1 dipende dal limite, e la codifica a quattro esiti non ha alcuna categoria per un compito lasciato cadere senza un rifiuto, che le esecuzioni agentiche possono produrre (§7).
- **Modello non fissato.** La CLI risolve il modello lato server, e all'interno di un singolo *batch* un secondo modello ha risposto a un'intera categoria. Ogni istantanea è legata ai modelli registrati nei suoi *envelope*, che non sono più disponibili per essere interrogati di nuovo.
- **Etichette, non esiti verificati**: *rifiutato* e *svolto* sono letti dal testo della risposta, perciò la §4 descrive queste etichette, non il successo del compito.
- **Un solo valutatore umano non in cieco**; il panel in cieco era composto da LLM della stessa famiglia del modello sotto test.
- **Ambiente non pulito** in entrambe le esecuzioni (§3.3).
- Il fenomeno e il consiglio di analizzare l'*envelope* **non sono inediti** (§2); il contributo è il conteggio misurato e le sue quattro letture, la misura dell'informazione della §4, l'argomento del contratto tipizzato della §5 e una verifica eseguita della ricetta difensiva.

Direzioni future, nell'ordine in cui intendiamo eseguirle:

(i) una terza istantanea secondo il protocollo del PREREGISTRATION.md del repository, depositato con un *timestamp* pubblico prima della raccolta: modello fissato, limiti di turni di 1 e 5, una precondizione di ambiente pulito (*clean-room*), un'etichetta del *codebook* di due valutatori in cieco come esito primario, e il rilevatore bloccato mantenuto come misura secondaria al limite di 1 turno per la confrontabilità;

(ii) un secondo valutatore umano sulle etichette di luglio, con il κ di Cohen (accordo tra due valutatori, al netto del caso) rispetto all'autore, usando il *codebook* sottoposto a *commit*;

(iii) precisione e *recall* della ricetta difensiva della §6 su un insieme etichettato riservato, includendo rifiuti non in inglese e una categoria *ignorato* per le esecuzioni con strumenti abilitati.

9. Disponibilità di dati e codice

Il materiale alla base di ogni numero di questa nota si trova nel repository pubblico anticipating-shadow-points, all'indirizzo https://github.com/ulissesflores/anticipating-shadow-points (Flores, 2026b). La Tabella 4 elenca ciò che contiene.

Tabella 4
Ciò che il repository contiene per questa nota

Materiale	**Dove nel repository**	**Che cosa può verificare il lettore**
L'esecuzione di maggio 2026: i 50 *prompt* e le 50 risposte	tests/iron-law-11/runs/published/	Le risposte e i verdetti del rilevatore di frasi alla base della §3.2
L'esecuzione di luglio 2026: le 50 risposte, con un indirizzo e-mail e un percorso locale rimossi da due risposte (i *prompt* sono quelli della suite bloccata, in tests/iron-law-11/scenarios/, abbinati per id dello scenario)	data/spike-2026-07-08/	Le risposte, le etichette dell'autore e il *codebook* alla base della §3.1, e un'impronta SHA-256 di ciascun record originale
Il calcolo	analysis/results.py	Come si calcola ogni conteggio, tasso, intervallo e quota di informazione, Tabella 3 compresa
I controlli	tests/test_paper_numbers.py	Che numeri selezionati corrispondono ai risultati ricalcolati dai dati; un secondo file, tests/test_paper_text.py, controlla che alcuni di essi compaiano nel testo di questa nota, e nessuno dei due controlla ogni numero che vi compare
Il sigillo	output/hash-chain.md	Una sola impronta SHA-256 su codice, dati e risultati, così che ogni modifica successiva si veda

Un’impronta SHA-256 è un codice di 64 caratteri calcolato a partire da un file; cambiare un solo byte del file cambia il codice. Un unico comando, python run_all.py, eseguito dentro la cartella del repository, rifà tutto ciò che è nella Tabella 4 e non chiama nessun modello; richiede Python 3.11 o successivo, e il README del repository elenca i quattro comandi che scaricano il repository, installano ciò che serve e lo eseguono. I record originali di luglio non sono pubblicati perché contengono anche percorsi di file e strumenti installati del computer dell’autore; le loro impronte, conservate nei file di luglio, permettono a chi li possiede di confermare che sono gli stessi record.

10. Dichiarazioni

Conflitto di interessi. L'autore mantiene ASP, un *plugin open source* per Claude Code, lo strumento misurato qui, e dirige il laboratorio che pubblica questa nota.

Finanziamento. Questo lavoro non ha ricevuto finanziamenti esterni. I costi d'uso dei modelli sono stati pagati dall'autore.

Contributo dell'autore. Autore unico: ideazione, disegno, raccolta dei dati, analisi, stesura e revisione.

Uso di strumenti di IA. I modelli linguistici hanno due ruoli in questo studio, entrambi descritti nella Sezione 3: l'agente provato e i cinque valutatori in cieco. Un assistente di IA è stato usato anche nella programmazione, nell'analisi e nella stesura e traduzione del testo, sotto la direzione dell'autore. L'autore ha verificato ogni numero sui dati e risponde del contenuto.

Riferimenti bibliografici

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 Tecnica

Versione 2.4 · novembre 2026

Serie Codex Hash Research Report 2026-02

Licenza Testo, figure e dati: CC BY 4.0. Software complementare: MIT.

Il progetto ASP. ASP (Anticipating Shadow Points) è un *plugin open source* per Claude Code, l'agente di programmazione da riga di comando di Anthropic, che fa elencare all'agente ciò che potrebbe andare storto prima di scrivere codice. Il *plugin* e il materiale di replicazione stanno in un repository pubblico (Flores, 2026b), un *whitepaper* descrive come funziona (Flores, 2026a) e questa nota misura da sola una delle sue regole: non fidarsi mai del codice di uscita di claude -p. La Sezione 5.5 del *whitepaper* riporta la prima misurazione di se ASP migliora i piani di un agente (nessun beneficio misurabile rispetto a una richiesta di una frase di elencare cosa può andare storto); un terzo articolo riporterà lo studio confermativo di quel pre-mortem di una frase.

DOI. Un DOI (Digital Object Identifier) è un collegamento permanente a un'opera depositata. Zenodo, un archivio pubblico di ricerca gestito dal CERN, conserva questa nota come record a sé: https://doi.org/10.5281/zenodo.24065934, che apre sempre la versione più recente; questa versione, la 2.4, è https://doi.org/10.5281/zenodo.24410426. Il *plugin*, i dati e il codice formano un altro record (Flores, 2026b), il cui DOI concept, https://doi.org/10.5281/zenodo.20276631, apre sempre la *release* più recente; la *release* 2.0.0 è quella che corrisponde a questa nota. Le *release* di maggio 2026 di quel record (da 1.0.0 a 1.0.3) chiamavano preregistrata l'esecuzione di maggio e precedono i dati di luglio; la *release* 2.0.0 le sostituisce.

Replicazione. Ogni numero che questa nota calcola dai suoi dati può essere ricalcolato dal repository senza chiamare un modello; la sezione 9, Disponibilità di dati e codice, elenca cosa contiene e come eseguirlo.

Sull'autore

Carlos Ulisses Flores è CTO e Ricercatore Capo presso il Codex Hash Research Laboratory, un laboratorio indipendente di R&S a San Paolo, in Brasile. Costruisce sistemi di produzione in cui il fallimento silenzioso è costoso — storicamente nel trading algoritmico, nell'IoT embedded e nella crittografia edge, oggi negli strumenti di IA agentica. Questa nota è nata dall'uso di claude -p come nucleo di esecuzione autonoma e dall'averlo visto restituire *exit code* 0 davanti a un rifiuto garbato.

Identificatori · ORCID: 0000-0002-6034-7765 · Lattes (registro nazionale brasiliano dei curricula dei ricercatori): 6905246706890561

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