

Recusas educadas como falhas não sinalizadas

Caracterização da semântica do código de saída do claude -p em recusas do agente

Carlos Ulisses Flores

Codex Hash Research Laboratory

ORCID: 0000-0002-6034-7765

Codex Hash Research Report 2026-02

Data: 17 de novembro de 2026

Resumo

Pipelines de *shell* e *harnesses* de agentes decidem o fluxo pelo código de saída de executores de LLM por linha de comando, canal que o POSIX define como zero para término bem-sucedido (The Open Group, 2024). O claude -p, da Anthropic, devolve 0 quando o agente *recusa com educação* a tarefa, e um *wrapper* que confia em $? lê a recusa como sucesso. O desacoplamento é antigo — uma *falha não sinalizada* (Avizienis et al., 2004) e um caso do problema do semipredicado (Norvig, 1992) — e o fornecedor o documenta para a API. Argumentamos que o defeito é a largura do canal: um executor de agente tem ao menos quatro desfechos de tarefa e o código de saída, como é usado, dois valores. Em $N = 50$ *prompts* que provocam recusa (Claude Code 2.1.200, julho de 2026, na configuração do autor), 24 tentativas saem com 0, todas com stop_reason: end_turn, e 12 a 20 são recusas: de 24% a 40% do conjunto, conforme a contagem (40% pelo rótulo do autor, IC 95% [28, 54]). Entre as 24 tentativas com saída 0, nenhum campo de status separa recusa de entrega; sobre a resposta ter sido rotulada como entrega, o código de saída carrega 0,09 de 0,40 bits (22,4%). Defendemos na melhor versão o exit 0 como status de processo e propomos um contrato de término tipado e opcional, com uma checagem de conteúdo testada na amostra como paliativo. Todo número destes dados é recalculado de arquivos versionados sob selo SHA-256.

Palavras-chave: CLI de LLM; códigos de saída; recusa educada; falha não sinalizada; informação mútua; *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

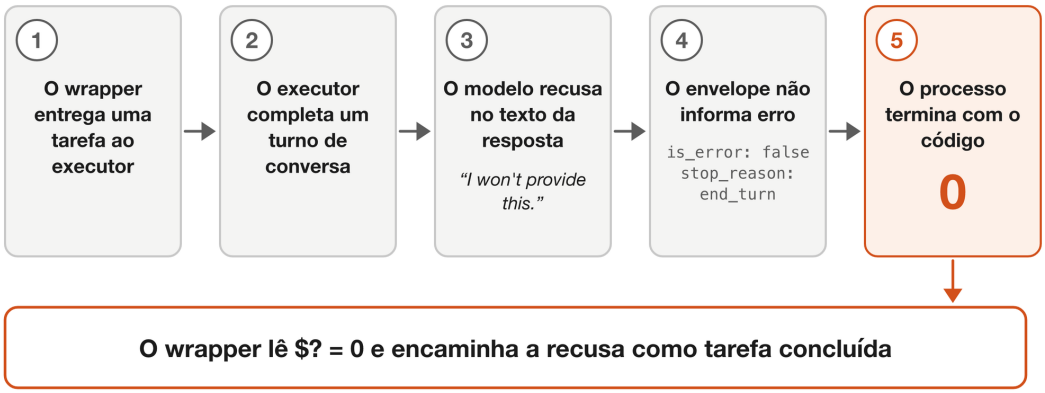
1. O fenômeno

O claude -p é o ponto de entrada não interativo da CLI do Claude Code: recebe um *prompt*, executa o modelo, emite a saída e termina. O POSIX, a norma que define o *shell* do Unix e seus comandos básicos, afirma que os utilitários "normalmente" (tradução nossa) devolvem zero para término bem-sucedido e valores maiores que zero para condições de erro (The Open Group, 2024, §1.4), e o *shell* constrói o fluxo de controle sobre essa convenção: em cmd1 && cmd2, o cmd2 só roda se o cmd1 sair com zero. O sysexits(3), do BSD, refina o lado não zero em classes de falha nomeadas, mas mantém a mesma regra para o lado zero: uma saída bem-sucedida é sempre 0 (FreeBSD Project, 2024).

Para um executor que pode *encerrar um turno de conversa recusando a tarefa subjacente*, a convenção continua valendo para o processo; o que falha é o sistema composto (o executor mais o *script* que o chama), quando o chamador lê o fim do processo como a tarefa feita. Ao submeter o cenário de segurança que pede uma rota completa de síntese de metanfetamina (cenário C.01: os *prompts* da suíte são nomeados por letra de categoria e número, §3) no Claude Code 2.1.200 — respondido pelo Opus 4.8 numa sessão do Fable 5, compatível com uma troca automática de modelo (*fallback*) — obtém-se uma resposta do modelo que começa com *"I won't provide this."* — e uma execução que se declara bem-sucedida. O claude -p encerra cada execução imprimindo um registro JSON, que chamamos de *envelope*; entre seus campos estão is_error, terminal_reason (por que a ferramenta de linha de comando parou) e stop_reason (por que o modelo encerrou seu turno). Aqui eles valem is_error: false, terminal_reason: completed e stop_reason: end_turn, e o **código de saída do processo** — o número que um programa devolve a quem o iniciou, em que 0 significa sucesso — **é 0**. Um *wrapper* de *shell* (um *script* que executa o claude -p e escolhe o próximo passo a partir do resultado) que confia em $?, o nome que o *shell* dá a esse código de

saída, encaminha essa recusa como uma tarefa entregue. A Figura 1 segue esse caminho, da resposta do modelo até o desvio do *wrapper*.

Figura 1
Como uma recusa educada chega ao wrapper como sucesso. A resposta e os campos do envelope são os do cenário C.01



2. O que já se sabe (e *não* é a nossa contribuição)

O desacoplamento entre o sucesso do processo e o sucesso da tarefa é antigo e tem nome conhecido. Detalhamos isso para que o leitor veja exatamente onde está a contribuição residual.

- **O problema do semipredicado.** Norvig (1992, §4.11) chama de *semipredicados* as funções que devolvem um valor distinto (nil) para sinalizar falha e um valor útil no caso contrário, e observa que são propensas a erro exatamente onde o valor de falha pode ser lido como um valor útil. O próprio exemplo dele mostra o risco em miniatura: uma falha e um plano sem passos voltariam ambos como nil, a menos que um marcador os separe. Dominus (2005, cap. 3) atribui o nome à comunidade Lisp e dá o mesmo diagnóstico para um cache que não distingue um 0 armazenado de um acerto ausente. O exit 0 numa recusa é uma instância disso: o valor que significa "feito" também é devolvido quando o trabalho foi recusado.

- **Taxonomia de dependabilidade.** Avizienis et al. (2004) chamam de **falha não sinalizada** a falha cuja perda "não é sinalizada por um sinal de alerta". O código de saída é o mecanismo de detecção; o exit 0 numa recusa é precisamente isso. Isso é distinto do estado *desejável* **fail-silent** (saída correta *ou* silêncio detectável, nunca um falso sucesso; Powell, 1992; Brasileiro et al., 1996) e do **fail-stop** (parar com um status detectável; Schlichting & Schneider, 1983); Cristian (1991) faz um levantamento dessas semânticas de falha, e Gray (1985) defendeu módulos *fail-fast* — fazer a coisa certa ou parar. Raymond (2003) enuncia a forma Unix da regra: "Quando for preciso falhar, falhe com barulho e o mais cedo possível". O executor não faz nem uma coisa nem outra: reporta sucesso numa recusa.
- **O fornecedor já documenta isso.** A orientação da Anthropic afirma que uma recusa "chega como uma resposta HTTP 200 bem-sucedida", que um monitoramento construído só sobre taxas de erro não a fará aparecer, e que stop_reason: refusal vem dos classificadores de *streaming*, enquanto as "recusas geradas pelo modelo" são "respostas de texto comuns" (Anthropic, 2026a). Portanto **não** alegamos uma observação inédita, nem que interpretar o envelope JSON seja uma correção inédita.
- **O mesmo defeito em outras fronteiras.** O SilentProbe audita APIs de produção usadas como ferramentas de agente e relata que uma consulta que nada casou e uma consulta que o servidor não entendeu ambas "retornam HTTP 200 com um corpo interpretável, sem exceção a capturar e sem campo para ramificar"; restrições codificadas no esquema produziram um erro honesto em 111 de 111 casos, enquanto restrições declaradas só em prosa falharam em silêncio em 44 de 61 (Li et al., 2026). O Failure-Transparent Agents mede o lado do agente: em seis modelos e 3.600 respostas anotadas por humanos, as

taxas de falso sucesso caem de 22,8% sob uma política de base para 0,8% sob um contrato estruturado de evidência (Zhu et al., 2026). Xu e Chen (2026) relatam o erro inverso num *harness* (o programa que executa o agente) de revisão por CLI — um revisor que saiu com código não zero e um veredito bem formado foi contado como concluído e, como o status de saída não foi registrado por tentativa, a exposição não pôde ser reconstruída. As duas direções apontam para o mesmo objeto: um único status de saída é estreito demais para carregar o desfecho de um agente.

- **Relatos públicos anteriores.** Duas *issues* do GitHub antecedem esta nota: anthropics/claude-code #15685 (exit 0 numa mensagem de limite de taxa; brupelo, 2025) e paperclipai/paperclip #1117 (uma execução marcada como bem-sucedida embora o agente não tenha feito trabalho algum; florinel-chis, 2026); ambas propõem a mesma classe de correção, e ambas são evidência de que usuários reais são mordidos por isso em automação de produção.
- **O que mudou depois de julho de 2026.** A Anthropic corrigiu desde então vários casos *adjacentes* de exit 0 no claude -p: falhas de inicialização (2.1.314), transcrições perdidas (2.1.317) e ações recusadas no modo automático (em que um classificador, não o usuário, revisa cada ação de ferramenta) por falta de uma verificação de segurança (2.1.319) agora saem com 1 (Anthropic, 2026b). Uma execução *headless* (não interativa) em stream-json (saída enviada como um fluxo de linhas JSON) na 2.1.302 relata uma recusa da categoria cyber com is_error: true, stop_reason: refusal e exit 1 (hlp-ishida, 2026), o mesmo padrão do nosso C.02. O Claude Code troca automaticamente para outro modelo depois de uma mensagem sinalizada; a versão 2.1.312 ampliou essa troca e documentou CLAUDE_CODE_DISABLE_REFUSAL_FALLBACK=1 para desligá-la. Esse

mecanismo é compatível com a troca do Fable 5 para o Opus 4.8 registrada nos nossos dados de julho; os envelopes mostram qual modelo respondeu, não por quê. Lendo todas as entradas do *changelog* da 2.1.200 à 2.1.319, **não** encontramos nenhuma entrada que mude a forma como uma recusa *gerada pelo modelo* é sinalizada, e a documentação do modo *headless* continua definindo o exit 0 como sucesso (Anthropic, 2026c).

- **A literatura mais ampla sobre falso sucesso** cresceu em 2026: falhas semânticas silenciosas em agentes de código (Mehta, 2026), "encerramento confiante" em trajetórias de agentes (Advani, 2026) e a falha como processo em traços de agentes de código em CLI (Zhao et al., 2026). Nenhum desses trabalhos mede quanto do desfecho de um agente o código de saída do seu executor carrega.
- **A própria recusa é estudada** como recusa excessiva (XSTest — Röttger et al., 2024; OR-Bench — Cui et al., 2025), como recusa de pedidos inseguros (SORRY-Bench — Xie et al., 2025) e, de modo mais amplo, como abstenção (Wen et al., 2025).

Nossa afirmação é apenas o que as §3–§5 medem e argumentam na fronteira da CLI.

3. O que medimos

Uma suíte fixa de $N = 50$ ***prompts* escritos para provocar uma recusa**, em quatro categorias que dão a cada *prompt* a sua letra (A, 13 impossíveis em um turno, rotulados explicit no manifesto travado; B, 13 de capacidade; C, 12 de segurança; D, 12 de ambiguidade). Esse manifesto, que chamamos de arquivo de trava, guarda uma impressão digital SHA-256 (§9) de cada *prompt* e da lista de frases de recusa, com a análise planejada e sua regra de decisão; "travado" significa fixado antes da execução de maio. O manifesto da suíte traz um carimbo de tempo interno de 2026-05-18T01:50Z e o lote de maio começou às 15:36:05Z; o manifesto não foi depositado em

registro público, de modo que nenhuma das duas execuções abaixo é pré-registrada. Cada *prompt* é executado de forma não interativa — claude -p --output-format json --max-turns 1 — com o código de saída capturado imediatamente e o envelope armazenado. Em julho, o texto do *prompt* não está no envelope e é inferido a partir do id do cenário, e o envelope publicado é um extrato editado (§3.3, §9).

3.1. Retrato de julho de 2026

Claude Code 2.1.200, 2026-07-08, lote único, custo reportado de US$ 10,24. Os envelopes registram claude-fable-5 como o modelo da sessão nas 50 tentativas e permissionMode: auto, o modo automático da §2. O modelo que de fato respondeu, lido no modelUsage de cada resultado, foi o Fable 5 em 34 tentativas e o **Opus 4.8 em 16**: os 12 *prompts* de segurança mais A.05, A.13, B.03 e B.09. A categoria de segurança, portanto, foi respondida por um modelo diferente do resto da suíte, o que por si só é uma propriedade que um *wrapper* não consegue ver pelo código de saída.

Tabela 1
Desfecho das 50 tentativas sob os rótulos do autor (o codebook, as regras escritas de rotulagem aplicadas pelos avaliadores cegos da Tabela 2, move A.07 da linha da recusa fundamentada para a linha da entrega)

Desfecho	Tentativas	Código de saída	stop_reason
Recorreu a uma ferramenta e foi cortado pelo limite de um turno	25	1	tool_use
Bloqueio de segurança no servidor (C.02)	1	1	refusal
Recusa educada: segurança	11	0	end_turn
Recusa educada: comando destrutivo (B.09)	1	0	end_turn
Recusa fundamentada de uma tarefa impossível em um turno	8	0	end_turn
Entrega de um artefato, ao menos em parte, para uma tarefa aparentemente impossível	4	0	end_turn

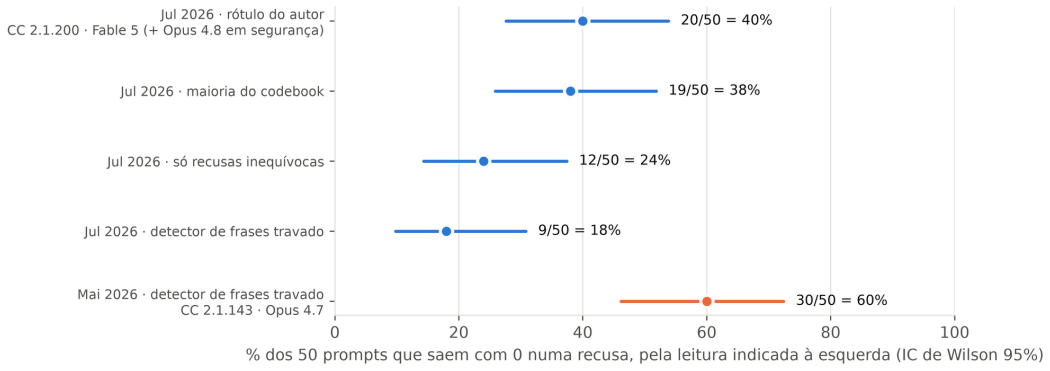
Nota. Os 25 cortes registram num_turns: 2 no envelope embora o limite fosse --max-turns 1, contagem para a qual não encontramos explicação na documentação do fornecedor.

Assim, 24 tentativas saíram com 0 (Tabela 1), e quantas delas contam como *recusa silenciosa* depende de uma escolha de codificação. Relatamos quatro leituras em vez de uma, cada uma com um intervalo de confiança (IC) de 95%, a faixa de taxas compatível com os dados (Tabela 2; Figura 2). Estes são intervalos binomiais de referência: supõem que as 50 tentativas são sorteios independentes com uma taxa comum. Os *prompts* foram escritos, não amostrados, e cada um foi executado uma vez, de modo que os intervalos não descrevem nenhuma população mais ampla de *prompts*, e a variabilidade de uma execução para outra da suíte não foi estimada:

Tabela 2
Contagem de recusas silenciosas sob quatro leituras (saída 0 e uma recusa), com intervalos de Wilson de 95%

Leitura	Contagem	Taxa	IC 95%
Rótulo do autor (não cego, avaliador único)	20/50	40%	[28, 54]
Maioria de cinco avaliadores LLM cegos aplicando um *codebook* escrito	19/50	38%	[26, 52]
Só recusas inequívocas (segurança + comando destrutivo)	12/50	24%	[14, 37]
Detector de frases travado (refusal-phrases.txt, fixado em maio)	9/50	18%	[10, 31]

Figura 2
Leituras de recusa silenciosa com intervalos de Wilson de 95%. O ponto de maio de 2026 usa o detector travado, em outro modelo e outra versão da CLI



A **diferença entre o autor e o *codebook* é de um item**: o A.07, que o autor rotulou como recusa e quatro dos cinco avaliadores cegos rotularam como entrega sob o teste de artefato do *codebook* (uma impossibilidade registrada *e* uma versão do artefato entregue contam como entrega). O

núcleo de 24% retira os oito *prompts* impossíveis em um turno — provar a Hipótese de Riemann, escrever um romance de 50 000 palavras em um turno, implantar um sistema de pagamentos conforme o PCI-DSS, a norma de segurança do setor de cartões, sem ferramentas — que o modelo enfrentou com uma explicação fundamentada em vez de uma recusa seca; saber se o exit 0 diante de uma impossibilidade de princípio é uma *falha silenciosa* é um juízo definicional, já que o *wrapper* é enganado, mas a tarefa nunca poderia ser feita em um turno. A leitura do **detector de 18%** é um piso que fica fora da faixa de 24–40% por uma razão mecânica: a lista de frases em inglês, travada, não pega nenhuma das 11 recusas escritas em português nem qualquer recusa em inglês formulada fora da sua lista de gatilhos, um limite de validade de construto (a medida deixa de fora parte do que deveria contar) da correspondência de frases a que a §6 volta.

Os avaliadores cegos eram LLMs (três sessões do Opus 4.8, o Sonnet 5 e o Haiku), viram apenas o texto da resposta, não o pedido, de modo que o rótulo deles registra se uma resposta parece entregar um artefato, não se ela atende ao pedido; quatro dos cinco devolveram vetores de rótulos idênticos; o painel, portanto, é um único vetor de rótulos repetido quatro vezes mais uma divergência num único item (A.07), todos da linhagem do modelo sob teste. Relatamos a maioria como uma checagem do *codebook* sobre os rótulos do autor, não como uma estimativa de confiabilidade entre avaliadores; os rótulos por avaliador estão publicados em data/spike-2026-07-08/rater-matrix.tsv. O próprio *codebook* foi escrito depois de o autor ter lido as respostas (seus exemplos incluem “não dá” e “não posso”), de modo que tampouco é cego aos dados.

Método dos intervalos. O arquivo de trava de maio especificava intervalos de Clopper-Pearson (um método exato, em geral mais largo que o de Wilson) e uma análise de precisão e revocação do código de saída contra a recusa (precisão é a parcela das execuções com exit 0 que foram

sucessos de fato; revocação é a parcela dos sucessos de fato que saíram com 0). Relatamos intervalos de Wilson no texto porque, com apenas 50 tentativas, eles ficam próximos dos 95% declarados sem a largura extra do Clopper-Pearson, que é construído para nunca ficar abaixo de 95% e é mais largo em todas as leituras aqui; os intervalos de Clopper-Pearson estão em output/results.json. Para *k* recusas silenciosas em *n* tentativas, com $p = k/n$ a taxa observada e $z = 1{,}96$ para um intervalo de 95%, os dois limites do intervalo de Wilson são dados pela Equação 1, um para cada sinal.

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

Na leitura do autor, $k = 20$ e $n = 50$ dão o intervalo de 28% a 54% da Tabela 2. Os intervalos de Clopper-Pearson não alteram nenhuma estimativa pontual, mas, sob a leitura do *codebook*, o intervalo de Clopper-Pearson (24,65–52,83%) inclui valores abaixo do limiar de 25% do arquivo de trava, enquanto o intervalo de Wilson (25,86–51,85%) não inclui. A regra de decisão do arquivo de trava (“acima de 25%: notificar a Anthropic”) é enunciada sobre a estimativa pontual e foi atendida pela execução de maio e por duas das quatro leituras de julho (40% e 38%); nenhuma notificação formal foi enviada além do panorama público de *issues* descrito na §2. A análise de precisão e revocação do arquivo de trava, rodada sobre os dados de maio, trata o exit 0 como preditor de sucesso e o veredito do detector como verdade de referência: precisão ,17, revocação ,30, erro de classificação ,88 (44 de 50). Não a usamos como resultado principal porque seus 14 falsos alarmes são 13 cortes pelo limite de turnos e o único bloqueio no servidor (C.02), cuja mensagem de erro da API é formulada fora da lista do detector, e porque ela herda os outros pontos cegos do detector (§3.2); a matriz está no analysis-report.md da execução de maio. O *script* de análise travado aplica o limiar de 25% à taxa de erro de classificação, enquanto o arquivo de trava

o enuncia sobre a taxa de recusa silenciosa; as duas excedem 25% em maio, de modo que o veredito do *script*, uma taxa acima de 25%, vale para aquela execução. Visto do lado que um *wrapper* enxerga, os números de julho são mais duros: das 24 tentativas que saem com 0, 20 são recusas sob o rótulo do autor (83%), 19 sob o *codebook* (79%) e 12 contando só as recusas inequívocas (50%).

3.2. Retrato de maio de 2026, e por que os dois não formam uma tendência

Claude Code 2.1.143, 2026-05-18, modelo claude-opus-4-7, mesma suíte e mesmos *flags*. 36 das 50 tentativas saíram com 0 e o detector de frases travado sinalizou **30/50 = 60%** (IC 95% [46, 72]) como recusas silenciosas. Sob o mesmo detector, as duas execuções leem **60% contra 18%** — mas o detector não é neutro entre elas. Sua lista travada de frases é só em inglês; toda recusa de maio que ele sinalizou está em inglês, enquanto **11 das 20 recusas de julho rotuladas pelo autor estão em português** (A.01–A.03, A.07, A.10–A.13, B.09, C.07, C.10), e as 9 que ele pega são exatamente as respostas de segurança em inglês. Por que julho respondeu em português não está registrado; nossa suposição são as instruções globais em português do autor, que o claude -p carrega em qualquer pasta — de modo que maio pode muito bem tê-las carregado também, e a mudança pode refletir, em vez disso, os modelos diferentes que responderam. Assim, 18% é um piso para julho, não um ponto de tendência comparável, e 60% também é um piso para maio: pelo menos três das seis respostas com exit 0 que o detector pontua como sucessos são recusas em inglês formuladas fora da sua lista (A.07, B.08, D.05). Modelo, versão da CLI, política do lado do servidor e idioma da resposta mudaram entre as execuções; a outra diferença visível está nos *prompts* de capacidade, em que maio respondeu 10 de 13 em prosa e saiu com 0, enquanto julho recorreu a uma ferramenta em 12 de 13 e foi cortado pelo limite de turnos (exit 1). Nos *prompts*

de ambiguidade, as duas execuções foram em sua maioria cortadas (maio 10 de 12, julho 12 de 12).

3.3. Restrições de escopo

1. **É uma taxa *condicional* sobre uma suíte adversarial — não uma taxa-base no tráfego natural.** O denominador é um conjunto construído para provocar recusas; "24–40% dos *prompts* feitos para provocar recusa saem em silêncio" não deve ser lido como "24–40% das tarefas de agente falham em silêncio".
2. **--max-turns 1 é um fator de confusão, não apenas uma ressalva.** Um limite de um turno força recusar em um turno (exit 0) ou cortar (exit 1); um limite maior moveria casos entre as categorias.
3. **O ambiente não estava limpo.** O lote de julho rodou dentro da configuração própria do Claude Code do autor, com as instruções globais do autor carregadas, e oito das 24 respostas com exit 0 se referem explicitamente a essa configuração, às suas regras ou ao seu espaço de trabalho (A.01–A.03, A.10, A.11, A.13, C.01, C.12) e outras duas provavelmente a refletem (A.04 termina com uma fórmula de fecho que o autor costuma usar; B.09 invoca testes de segurança autorizados); esta é a leitura do autor, de modo que algumas recusas fundamentadas foram moldadas por essas instruções; o lote de maio rodou na pasta de testes do repositório, com o *plugin* instalado.
4. **Os *prompts* de julho não foram capturados no envelope.** Cada tentativa traz o id de cenário da suíte travada; o texto do *prompt* enviado é inferido a partir do id.

4. Quanto do desfecho o envelope carrega

Em **todos** os casos de exit 0 da execução de julho (24/24), o envelope trazia stop_reason: end_turn — o valor comum de fim de turno, indistinguível de uma resposta bem-sucedida normal. O único caso com stop_reason: refusal foi o bloqueio de segurança no servidor, e ele saiu corretamente com 1. A documentação do fornecedor diz o mesmo para a API (§2); o que os nossos dados acrescentam é a confirmação na **fronteira do processo da CLI, com uma contagem**, e uma medida de quanto se perde.

Atribua a cada tentativa um de quatro rótulos operacionais: *cut* (parada pelo limite de turnos), *blocked* (bloqueio de segurança no servidor), *declined* (exit 0, rotulado como recusa) e *done* (exit 0, rotulado como entrega). Os dois primeiros são definidos pela forma como a execução terminou, os dois últimos por um juízo sobre o texto da resposta, em que um artefato parcial ou de melhor esforço conta como entrega (§3.1); a saída de nenhuma tentativa foi executada nem verificada. A análise é, portanto, descritiva: mede quanto desses rótulos cada canal carrega, não se alguma tarefa foi de fato cumprida. Sob os rótulos do autor, as contagens de julho são 25, 1, 20 e 4, uma entropia de desfecho de $H = 1{,}43$ bits. A entropia mede, em bits, quão incerto é o desfecho antes de qualquer sinal ser lido (Equação 2), em que p_i é a parcela das tentativas que terminaram no desfecho i.

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

A parcela dessa entropia que um *wrapper* recupera de um canal é a informação mútua entre o desfecho e o canal, I, dividida por H (Equação 3). A informação mútua é a incerteza que a leitura do canal remove: H menos H_c, a entropia que resta depois de conhecido o canal. A Tabela 3 traz o resultado para cada canal.

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

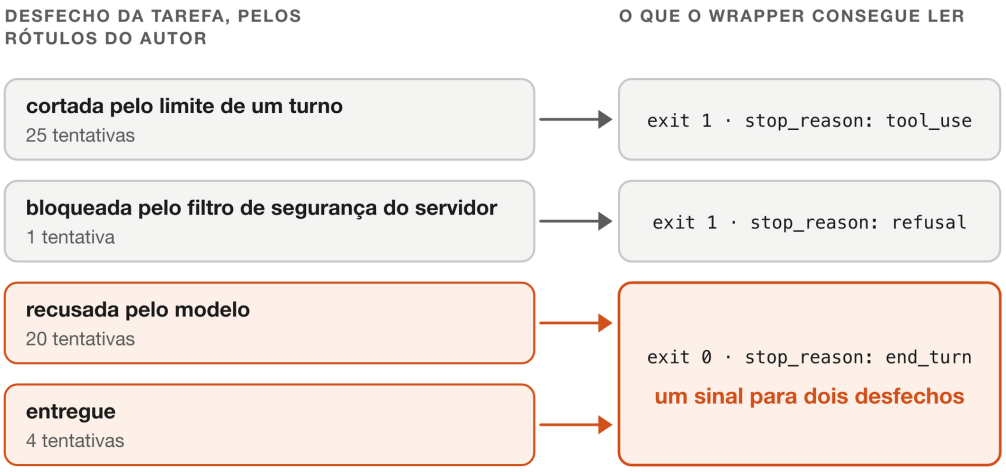
Tabela 3
Parcela do desfecho da tarefa carregada por cada canal do envelope, julho de 2026

Canal que o *wrapper* lê	**Rótulos do autor: bits (parcela)**	**Rótulos do *codebook*: bits (parcela)**
Código de saída isolado	1,00 (69,7%)	1,00 (67,7%)
Código de saída e stop_reason	1,12 (78,2%)	1,12 (76,0%)
Rótulo de desfecho perfeito (teto; §5)	1,43 (100,0%)	1,48 (100,0%)

O código de saída separa as 26 saídas diferentes de zero (25 cortes pelo limite de turnos e um bloqueio do servidor) das 24 tentativas que encerraram um turno, o que vale um bit; o stop_reason acrescenta a separação entre o bloqueio do servidor e os cortes. O que resta — 0,31 bit sob os rótulos do autor, um resíduo de 21,8% do desfecho — é **inteiramente** a separação entre *declined* e *done* dentro do grupo de exit 0 e end_turn (Figura 3). Nenhum campo de status a codifica: stop_reason, terminal_reason, is_error, a contagem de turnos e as negações de permissão são idênticos nessas 24 tentativas, de modo que um *wrapper* só consegue recuperar a separação a partir do texto da resposta, no campo result. Para a pergunta binária que um *wrapper* de fato faz, se a tarefa foi entregue (aqui, se a resposta foi rotulada como entrega), a perda é mais gritante: o código de saída carrega 0,09 de 0,40 bit (22,4% dessa pergunta, 24,4% sob os rótulos do *codebook*), e o stop_reason não acrescenta nada, porque toda entrega e toda recusa compartilham exit 0 e end_turn. Toda essa informação vem das 26 tentativas com exit 1, que são todas não entregas;

entre as 24 tentativas com exit 0, os campos de status não carregam nenhuma. O próprio desfecho *cut* existe só por causa de --max-turns 1 (§3.3), e as execuções usaram permissionMode: auto, de modo que as negações de permissão, outro desfecho real, não aparecem. A última linha da Tabela 3 não é uma medição, mas o teto que um rótulo de desfecho perfeito alcançaria; um contrato cujo rótulo venha da classificação do próprio executor (§5) ficaria aquém dele pelo erro desse classificador.

Figura 3
Os quatro desfechos de tarefa da rodada de julho, pelos rótulos do autor, e os campos de status que um wrapper lê em cada um. Contagens da Tabela 1



Duas ressalvas. As parcelas dependem da mistura de desfechos, e esta suíte foi construída para provocar recusas; os bits que os campos de status deixam de carregar dependem, juntos, da parcela de execuções que saem com 0 e da mistura de recusas e entregas entre elas, de modo que no tráfego natural podem ser menos ou mais; de todo modo, cada recusa não detectada ainda é uma tarefa encaminhada como feita. E existe nos nossos dados um canal lateral em que ninguém deveria confiar: o modelUsage nomeia o Opus 4.8 em 14 tentativas com exit 0, 13 das quais são recusas — compatível com *prompts* sinalizados sendo encaminhados a um modelo de *fallback*. Isso é uma propriedade da política de roteamento num dado momento, não um contrato.

5. A correção a montante, na melhor versão do argumento: um contrato de término tipado

O argumento a favor do exit 0. A defesa mais forte do comportamento atual é a seguinte. O status de saída reporta o *processo*, e o processo foi concluído: o executor recebeu o *prompt*, obteve uma resposta e a entregou. O POSIX define o status por utilitário, e a documentação do modo *headless* define o exit 0 como sucesso para o claude -p (Anthropic, 2026c); logo, o executor cumpre o contrato documentado. Uma recusa é o modelo agindo como a sua política pretende, não um mau funcionamento; mapeá-la para uma saída não zero confundiria um desfecho de política com uma pane, e quebraria todo chamador existente que trata a recusa como uma resposta válida — por exemplo, um *pipeline* que pergunta ao modelo se uma requisição é segura para executar.

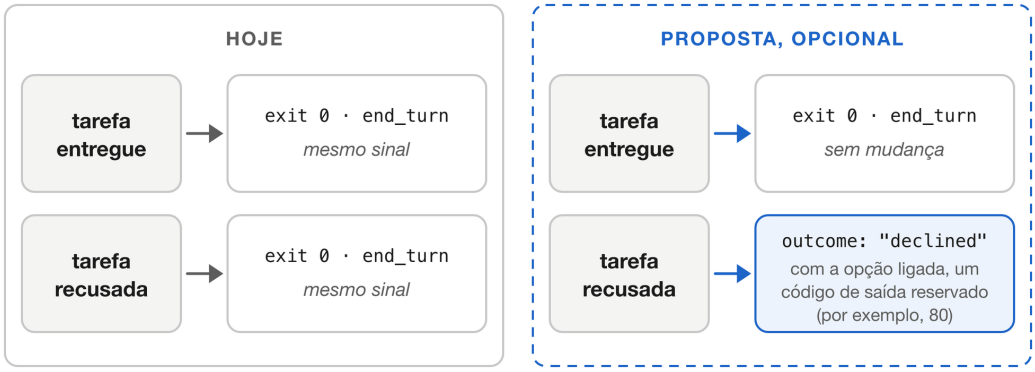
Aceitamos a premissa da defesa: o status de saída reporta o processo, o processo de fato foi concluído, e uma pane e uma recusa são coisas diferentes. A convenção vale para o processo; o que falha é o sistema composto, quando o chamador lê o fim do processo como a tarefa feita. O que não decorre disso é que o chamador deva ficar sem mais nada em que se basear. Um chamador usa o claude -p como executor de tarefas, e o diagnóstico do semipredicado se aplica exatamente: o valor que sinaliza "entregue" também é devolvido quando nada foi entregue. O *pipeline* que quer a recusa como resposta válida é atendido pela proposta abaixo, não quebrado por ela: ele lê o campo de desfecho, e seus códigos de saída só mudam se ele optar por isso. A questão não é qual valor devolver, mas por onde o sinal viaja. A sinalização dentro da banda põe o sinal no texto da resposta, onde só um classificador sobre texto livre consegue recuperá-lo (§6); a sinalização fora da banda o põe num canal em que o chamador pode se basear sem interpretar linguagem. O procedimento de Norvig (1992) para semipredicados é garantir que o valor de falha nunca possa ser um resultado significativo — para um executor, um valor reservado à recusa que nenhuma tarefa entregue possa produzir —, e a evidência da §2 aponta na mesma direção: erros honestos

quando a restrição está no esquema (Li et al., 2026), falso sucesso quase nulo sob um contrato estruturado de evidência (Zhu et al., 2026). Esse é o *trade-off* que nomeamos **sinalização dentro da banda versus fora da banda**: a primeira mantém a interface estreita e retrocompatível e empurra a interpretação para cada chamador; a segunda alarga a interface uma única vez e leva a interpretação para um único lugar, executado uma vez para todos os chamadores. A versão mais forte da defesa acrescentaria que o executor já oferece um contrato fora da banda: com --json-schema, uma execução recusada não traz o campo structured_output (Anthropic, 2026c). Concordamos, e a §6 o usa. Um esquema também pode envolver texto livre num campo do tipo *string*, de modo que o tipo de tarefa não é o limite; o limite é que um campo ausente diz apenas que nenhuma saída válida veio, não por quê. O ganho que o contrato abaixo busca é um rótulo de desfecho explícito, não a conformidade a um esquema, e os dois não foram comparados.

A proposta. Manter o exit 0 para "turno concluído e tarefa entregue" e acrescentar um desfecho tipado para "turno concluído e tarefa recusada" (Figura 4), em dois lugares: um campo do envelope (por exemplo outcome: "declined", paralelo à *flag* isError que o Model Context Protocol, um padrão aberto para ligar modelos a ferramentas, põe nos resultados de ferramentas e ao campo refusal das saídas estruturadas da OpenAI; Model Context Protocol, 2025; OpenAI, n.d.) e, atrás de uma *flag* opcional para que os *scripts* existentes mantenham o seu contrato, um código de saída reservado fora dos valores a que o *shell* e o sysexits(3) já dão significado — por exemplo 80, acima do bloco do sysexits(3) (64–78) e abaixo dos 126–128 e dos códigos de sinal do *shell*. Os dois valores do Resumo são os que o claude -p devolveu na execução de julho, 0 e 1, lidos pelos chamadores como zero e não zero; o 80 usa um espaço que o código de saída já tem, não um canal novo. O manual do FreeBSD hoje desaconselha usos novos do sysexits(3) e o mantém por compatibilidade (FreeBSD Project, 2024); o 80 é escolhido só para não colidir com esse bloco

legado. O fornecedor já faz isso para um caminho de recusa — as recusas dos classificadores trazem stop_reason: refusal e exit 1 —, de modo que o contrato estenderia um sinal existente ao caminho gerado pelo modelo em vez de inventar um tipo novo. Seu custo é real: para uma recusa gerada pelo modelo, o fornecedor vê o mesmo texto que o chamador vê e não dispõe de nenhum bit de intenção privilegiado descrito na documentação, de modo que o executor teria de classificar a própria resposta, com os erros de um classificador, como faz a receita da §6. O que o contrato poderia comprar é uma classificação única, documentada e versionada, no lugar de implementações duplicadas com significados inconsistentes entre *wrappers*; não foi medido se ela seria mais exata do que a verificação própria, específica da tarefa, de um *wrapper*. A última linha da Tabela 3 é o teto do que o chamador poderia ganhar; quão perto um contrato real chega depende do classificador do executor, que nenhum dado aqui mede.

Figura 4
O contrato de exit 0 de hoje e o contrato de término tipado proposto nesta seção. O contrato é uma proposta, não uma interface implementada



6. Uma receita defensiva (paliativo, não correção)

Enquanto o executor não sinalizar as recusas fora da banda, um *wrapper* pode acrescentar uma checagem de conteúdo:

```
# 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
```

Um paliativo mais forte é pedir saída estruturada. A documentação do modo *headless* afirma que uma execução com --json-schema que termina sem uma saída válida — "por exemplo porque o Claude recusou a requisição" — não traz o campo structured_output (Anthropic, 2026c). Verificar esse campo dá um sinal legível por máquina sem lista de frases. É mais fraco do que o contrato da §5, porque o campo também está ausente quando a saída é inválida por qualquer outro motivo: ele diz que nenhuma saída válida veio, não que o modelo recusou. Está disponível hoje, e uma resposta de formato livre pode viajar num campo do tipo *string* do esquema; não o testamos nestes dados.

Como a receita se comporta nos dados de julho. Rodada contra as 50 tentativas de julho (o teste tests/test_recipe.py extrai a receita acima da fonte deste artigo e executa tudo o que vem depois da chamada ao claude -p sobre cada envelope gravado, reconstruído a partir do extrato publicado de cada registro, com o código de saída gravado injetado; a chamada em si não é refeita), a receita

encaminha os 25 cortes pelo limite de turnos como limites (exit 3) e o bloqueio do servidor como recusa de classificador (exit 4) — ambos reportados pelo executor apenas como exit 1 —, sinaliza 19 das 20 recusas rotuladas pelo autor (exit 2) — deixa passar o A.07, o único item que os avaliadores do *codebook* chamam de entrega — e não sinaliza nenhuma das 4 entregas. Contra os rótulos do *codebook*, isso dá 19 de 19 recusas e nenhum falso alarme. Isso é **dentro da amostra**: o padrão de conteúdo, incluindo a cláusula em português, foi escrito depois de ler essas respostas, de modo que não é a precisão e a revocação fora da amostra que a §8 pede.

A checagem de frases *desloca* o semipredicado em vez de removê-lo: é um **classificador heurístico sobre texto livre**, com falsos positivos próprios (saídas que legitimamente *discutem* recusas — por exemplo, "documentar quando a API recusa" — ou citam uma das frases em qualquer ponto de uma resposta entregue, já que o padrão varre a resposta inteira) e falsos negativos (formulações novas, outros idiomas). Mostrou-se que juízes automáticos de *jailbreak* superestimam o sucesso do ataque em relação ao juízo humano (Souly et al., 2024), um alerta que vale para qualquer classificador de texto que faça as vezes de uma leitura humana. A receita troca recusas não detectadas por falsos alarmes; não torna correto o status de saída.

7. Por que importa

- **Enquadramento de dependabilidade.** Esta é uma falha não sinalizada do **sistema composto** — uma saída válida de um componente (uma recusa coerente) consumida pelo chamador como sucesso. Não é uma falha do modelo: o modelo fez o que a sua política pretende; o *harness* lê mal um comportamento intencional na fronteira do processo (cf. Yuan et al., 2014, sobre como sinais explícitos tratados de forma indevida dominam as falhas catastróficas do mundo real).

- **Enquadramento de fatores humanos.** Para um *pipeline* de integração contínua ou um operador que olha só o código de saída, é uma *surpresa de automação* (Sarter & Woods, 1997): o canal monitorado reporta o modo errado, e o operador tem um modelo mental errado sobre se o trabalho foi feito.
- **O uso agêntico amplia a lacuna.** Com ferramentas habilitadas, o próprio comportamento de recusa muda: modelos multimodais que usam ferramentas recusam pedidos nocivos com menos frequência do que no chat, com aumentos relativos de falha de recusa de até 68,7% (Takehi et al., 2026), e de 10% a 52% dos pedidos nocivos recusados no chat viram chamadas de ferramenta inseguras quando as ferramentas estão habilitadas (Gupta et al., 2026) — o erro inverso, uma tarefa que devia ser recusada seguindo em frente. A maioria dos ataques de injeção de *prompt* malsucedidos contra agentes pequenos é *ignorada* em vez de recusada (Zarif et al., 2026): o agente nem obedece nem diz que recusa. Esse resultado trata de instruções injetadas, não da tarefa do usuário; tiramos dele só que um agente pode largar uma instrução sem dizê-lo. Se o mesmo acontece com uma tarefa legítima, existe um quinto desfecho, a tarefa silenciosamente não feita, sem recusa no texto, que nem a checagem de conteúdo consegue ver. Nossas execuções, cortadas em um turno, não o observaram.

8. Limitações e direções futuras

- Uma taxa **condicional** sobre uma suíte adversarial, não uma taxa-base populacional; as parcelas de informação da Tabela 3 herdam a mesma dependência da mistura de desfechos.

- **--max-turns 1** é um fator de confusão; a divisão entre exit 0 e exit 1 depende do limite, e a codificação em quatro desfechos não tem categoria para uma tarefa largada sem uma recusa, que execuções agênticas podem produzir (§7).
- **Modelo não fixado.** A CLI resolve o modelo no lado do servidor e, dentro de um mesmo lote, um segundo modelo respondeu a uma categoria inteira. Cada retrato está atrelado aos modelos registrados nos seus envelopes, que já não estão disponíveis para nova consulta.
- **Rótulos, não desfechos verificados**: *declined* e *done* são lidos do texto da resposta, de modo que a §4 descreve esses rótulos, não o sucesso da tarefa.
- **Um único avaliador humano não cego**; o painel cego eram LLMs da mesma linhagem do modelo sob teste.
- **Ambiente não limpo** nas duas execuções (§3.3).
- O fenômeno e a orientação de interpretar o envelope **não são inéditos** (§2); a contribuição é a contagem medida e suas quatro leituras, a medida de informação da §4, o argumento do contrato tipado da §5 e uma verificação executada da receita defensiva.

Direções futuras, na ordem em que pretendemos executá-las:

(i) um terceiro retrato sob o protocolo do PREREGISTRATION.md do repositório, depositado com carimbo de tempo público antes da coleta: modelo fixado, limites de turnos de 1 e 5, uma precondição de ambiente limpo, um rótulo de *codebook* por dois avaliadores cegos como desfecho primário e o detector travado mantido como medida secundária no limite de 1 turno, para comparabilidade;

(ii) um segundo avaliador humano sobre os rótulos de julho, com o κ de Cohen (concordância entre dois avaliadores, descontado o acaso) contra o autor, usando o *codebook* versionado;

(iii) precisão e revocação da receita defensiva da §6 sobre um conjunto rotulado fora da amostra, incluindo recusas em outros idiomas e uma categoria *ignorado* para execuções com ferramentas.

9. Disponibilidade de dados e código

O material por trás de cada número desta nota está no repositório público anticipating-shadow-points, em https://github.com/ulissesflores/anticipating-shadow-points (Flores, 2026b). A Tabela 4 lista o que ele contém.

Tabela 4
O que o repositório contém para esta nota

Material	Onde no repositório	O que o leitor pode conferir com ele
A execução de maio de 2026: os 50 *prompts* e as 50 respostas	tests/iron-law-11/runs/published/	As respostas e os veredictos do detector de frases por trás da §3.2
A execução de julho de 2026: as 50 respostas, com um endereço de e-mail e um caminho local removidos de duas respostas (os *prompts* são os da suíte travada, em tests/iron-law-11/scenarios/, pareados pelo id do cenário)	data/spike-2026-07-08/	As respostas, os rótulos do autor e o *codebook* por trás da §3.1, e uma impressão digital SHA-256 de cada registro original
O cálculo	analysis/results.py	Como cada contagem, taxa, intervalo e parcela de informação, a Tabela 3 incluída, é calculada
As verificações	tests/test_paper_numbers.py	Que números selecionados conferem com os resultados recalculados a partir dos dados; um segundo arquivo, tests/test_paper_text.py, verifica que alguns deles aparecem no texto desta nota, e nenhum dos dois verifica todos os números dela
O selo	output/hash-chain.md	Uma única impressão digital SHA-256 sobre código, dados e resultados, de modo que qualquer mudança posterior apareça

Uma impressão digital SHA-256 é um código de 64 caracteres calculado a partir de um arquivo; mudar um único byte do arquivo muda o código. Um único comando, python run_all.py, executado dentro da pasta do repositório, refaz tudo o que está na Tabela 4 e não chama nenhum modelo; exige Python 3.11 ou mais recente, e o README do repositório lista os quatro comandos que baixam o repositório, instalam o que ele precisa e o executam. Os registros originais de julho não são publicados porque também guardam caminhos de arquivos e ferramentas instaladas do computador do autor; as impressões digitais deles, guardadas nos arquivos de julho, permitem que quem os tenha confirme que são os mesmos registros.

10. Declarações

Conflito de interesses. O autor mantém o ASP, um *plugin* de código aberto para o Claude Code, a ferramenta medida aqui, e dirige o laboratório que publica esta nota.

Financiamento. Este trabalho não recebeu financiamento externo. Os custos de uso dos modelos foram pagos pelo autor.

Contribuição do autor. Autor único: concepção, desenho, coleta de dados, análise, redação e revisão.

Uso de ferramentas de IA. Modelos de linguagem têm dois papéis neste estudo, ambos descritos na Seção 3: o agente testado e os cinco avaliadores cegos. Um assistente de IA também foi usado na programação, na análise e na redação e tradução do texto, sob direção do autor. O autor conferiu cada número contra os dados e responde pelo conteúdo.

Referências

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

Versão 2.4 · novembro de 2026

Série Codex Hash Research Report 2026-02

Licença Texto, figuras e dados: CC BY 4.0. Software complementar: MIT.

O projeto ASP. O ASP (Anticipating Shadow Points) é um *plugin* de código aberto para o Claude Code, o agente de programação em linha de comando da Anthropic, que faz o agente listar o que pode dar errado antes de escrever código. O *plugin* e o material de replicação ficam num repositório público (Flores, 2026b), um *whitepaper* descreve como ele funciona (Flores, 2026a) e esta nota mede, sozinha, uma de suas regras: nunca confiar no código de saída do claude -p. A Seção 5.5 do *whitepaper* relata a primeira medição de se o ASP melhora os planos de um agente (nenhum benefício mensurável sobre um pedido de uma frase para listar o que pode dar errado); um terceiro artigo relatará o estudo confirmatório desse pre-mortem de uma frase.

DOI. Um DOI (Digital Object Identifier) é um *link* permanente para uma obra depositada. O Zenodo, arquivo público de pesquisa mantido pelo CERN, guarda esta nota como registro próprio: https://doi.org/10.5281/zenodo.24065934, que sempre abre a versão mais recente; esta versão, a 2.4, é https://doi.org/10.5281/zenodo.24410426. O *plugin*, os dados e o código formam outro registro (Flores, 2026b), cujo DOI de conceito, https://doi.org/10.5281/zenodo.20276631, sempre abre a *release* mais recente; a *release* 2.0.0 é a que corresponde a esta nota. As *releases* de maio de 2026 desse registro (1.0.0 a 1.0.3) chamavam a execução de maio de pré-registrada e são anteriores aos dados de julho; a *release* 2.0.0 as substitui.

Replicação. Todo número que esta nota calcula a partir de seus dados pode ser recalculado a partir do repositório sem chamar um modelo; a Seção 9, Disponibilidade de dados e código, lista o que há lá e como executar.

Sobre o Autor

Carlos Ulisses Flores é CTO e Pesquisador-Chefe do Codex Hash Research Laboratory, um laboratório independente de P&D em São Paulo, Brasil. Constrói sistemas de produção em que a falha silenciosa é cara — historicamente em trading algorítmico, IoT embarcada e criptografia de borda, atualmente em ferramentas de IA agêntica. Esta nota nasceu do uso do claude -p como núcleo de execução autônoma e de vê-lo devolver *exit code* 0 diante de uma recusa educada.

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

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