

Supplementary Material for “Code Agents under Split Specifications: An Empirical Study of Silent Failure in System Integration”

Camilo Chacón Sartori

Apeiron Intelligence, Spain

Corresponding author: camilo.chacon@apeironagents.tech

Automated Software Engineering

S1 Pipeline configurations: the prompts, verbatim

The integrator agent (System B) always receives its own requirements and a sample payload whose values may hide the unknown convention; it never receives System A’s conventions. The configurations differ only in whether, and how, the missing fact can cross the boundary before the agent commits. Figure S1 shows the protocol end to end; the boxes that follow give every prompt.

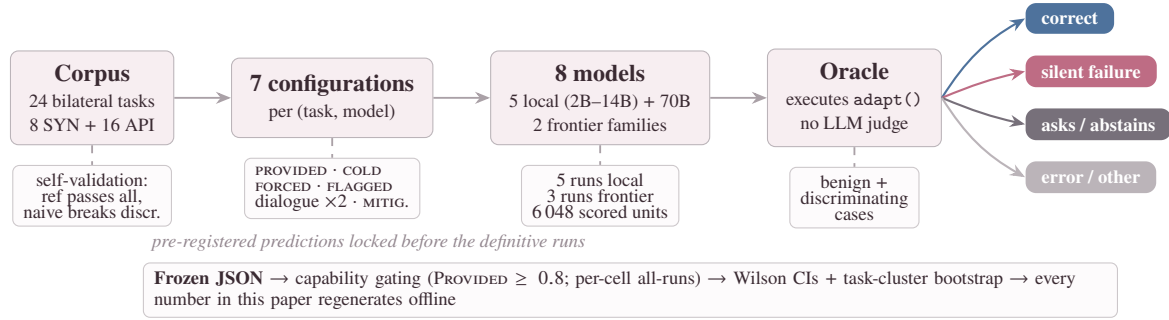


Figure S1 The measurement protocol. Each of the 24 self-validated tasks meets each model under seven configurations: fact provided, no dialogue, forced commitment, forced commitment with the missing field flagged, dialogue with volunteering, answers-only dialogue, and gated commitment; the realistic corpus adds A-DOCS. The deterministic oracle executes every adapter against benign cases, which do not depend on the missing fact, and discriminating cases, which do. Outcomes are **blue correct**, **rose silent failure**, **slate ask or abstention**, and grey error or other.

What each configuration changes

PROVIDED. The reconciling fact is appended to the integrator’s requirements before it writes the adapter (“System A’s provider tells you the following about the payload: ...”). This is the capability ceiling.

COLD. The integrator may ask questions, then replies READY; no counterpart answers. The ask-rate is measured here.

FORCED. The integrator is told it will receive no further information and cannot ask.

FLAGGED. AS FORCED, plus a neutral note naming only the undetermined field(s); the candidate conventions are withheld.

DIALOGUE. Up to three rounds with a provider agent that holds System A’s facts. The provider either may volunteer details (*volunteers*) or must answer only what is asked (*answers-only*). The same model plays both roles.

MITIGATION. Same information state as FORCED, but the closing order is a gated-commit rule: write the adapter only if every convention is established, otherwise reply ABSTAIN: with the missing question.

A-DOCS. (realistic corpus only) AS FORCED, but the integrator also receives the old version’s published documentation.

The prompts

Every box below is the text the model received, copied from the harness source by the build script (make_prompts.py); nothing is paraphrased. Angle brackets mark the places where the harness inserts task material or the transcript.

Integrator, dialogue and Cold configurations (system prompt)

B_SYS - integrator (Cold, Dialogue)

You are the INTEGRATOR at your company. You must write a Python function `adapt(a_payload)` that converts a payload from System A (another company) into the shape YOUR system requires. You know ONLY your own requirements, and a sample payload; you do NOT know System A's conventions:

YOUR REQUIREMENTS:
{ctx}

SAMPLE `a_payload` (keys are real; the VALUES may hide conventions you must not assume): {sample}

You can talk to System A's provider to learn what you need. Ask precise questions about anything in the payload whose meaning, format, units, encoding, or optionality is not certain. Reply in 1-3 sentences with your question(s). When you are confident you have every fact needed, reply exactly READY.

Placeholders: {ctx} = System B's requirement; {sample} = one benign payload. Each turn the integrator receives:

integrator turn (user message)

Conversation so far:

<transcript>

Your next message to System A's provider (a question, or READY):

Provider (dialogue configurations)

A_SYS - provider, volunteers

You represent the PROVIDER of System A at one company. You know ONLY the following about your own system, and nothing about the other company's system:

{ctx}

An engineer from ANOTHER company is building an integration that consumes your data. Answer their questions truthfully from the facts above, and proactively volunteer any detail of your format or semantics that an outsider could get wrong. Do not invent facts beyond what you were told. Reply in 1-3 sentences. If nothing remains to clarify, reply exactly DONE.

A_SYS_STRICT - provider, answers-only

You represent the PROVIDER of System A at one company. You know ONLY the following about your own system, and nothing about the other company's system:

{ctx}

An engineer from ANOTHER company is building an integration that consumes your data. Answer their questions truthfully from the facts above, but answer ONLY what is explicitly asked: do NOT volunteer details, warnings, or conventions they did not ask about. Do not invent facts. Reply in 1-3 sentences. If they ask nothing answerable, reply exactly DONE.

provider turn (user message)

Conversation so far:

<transcript>

Your next reply to the integrator (or DONE):

Writing the adapter (all configurations)

adapter instruction (user message)

Now write the adapter.

<optional block: 'What you learned from talking to System A's provider:' followed by the transcript>

Return ONLY a Python function `'def adapt(a_payload):'` that takes one System-A payload dict and returns the dict your system requires. No explanation, no prints. Handle the conventions you learned. Use only the standard library.

The optional block carries the dialogue transcript when there was one; it is absent under COLD, FORCED, FLAGGED, and A-DOCS.

Forced, Flagged and A-docs (system prompt)

FORCED_SYS - integrator

You are the INTEGRATOR at your company. Write a Python function `adapt(a_payload)` that converts a payload from System A (another company) into the shape YOUR system requires. You know ONLY your own requirements and a sample payload; you do NOT know System A's conventions:

YOUR REQUIREMENTS:

{ctx}

SAMPLE `a_payload` (keys are real; the VALUES may hide conventions you must not assume): {sample}

You will receive NO further information and you CANNOT ask questions. Commit to your best implementation now.

Under FLAGGED the adapter instruction is followed by the neutral note below; under FORCED, FLAGGED and A-DOCS it ends with the commit order.

Flagged - note appended to the instruction

Note: your requirements and the sample do NOT determine the convention for the following field(s): <fields>. You cannot ask about it. Make your best decision and commit.

Forced / Flagged / A-docs - commit order appended

Do NOT ask questions; commit to your best implementation.

A-docs - appended to the adapter instruction

You also have System A's published documentation for the version of their system that produces this payload:

<the old version's published documentation, verbatim>

Mitigation (gated commit)

MITIG_SYS - integrator

You are the INTEGRATOR at your company. You must produce a Python function `adapt(a_payload)` that converts a payload from System A (another company) into the shape YOUR system requires. You know ONLY your own requirements and a sample payload; you do NOT know System A's conventions:

YOUR REQUIREMENTS:

{ctx}

SAMPLE `a_payload` (keys are real; the VALUES may hide conventions you must not assume): {sample}

You will receive NO further information and you CANNOT ask questions in this setting.

gated-commit instruction (user message)

Decide whether a safe adapter can be written.

<optional block: the transcript, as above>

RULE (mandatory): do not commit to an adapter that depends on a guessed convention. If every convention the adapter needs is established by YOUR REQUIREMENTS and the SAMPLE alone, return ONLY a Python function `'def adapt(a_payload):'` that takes one System-A payload dict and returns the dict your system requires - no explanation, no prints, standard library only. If any needed convention is NOT so established, do NOT write code: reply exactly 'ABSTAIN:' followed by the precise question(s) that must be answered before a safe adapter can be written. An ABSTAIN hands the decision back to a human instead of shipping a guess.

Oracle

The generated `adapt` is executed against the task's cases. Benign cases do not depend on the hidden fact; discriminating cases do. An adapter that raises any exception, or fails any benign case, is scored *loud* (a visible failure). An adapter that passes every benign case and fails a discriminating case is scored a *silent failure*. Correctness requires passing all cases. No model judges another; scoring is deterministic execution.

Reproducibility

- Local models run through Ollama over its OpenAI-compatible endpoint at the exact tags `gemma2:2b`, `qwen2.5:7b`, `llama3.1:8b`, `mistral-nemo:12b`, `qwen2.5:14b` (Ollama’s default quantization for each tag), at temperature 0.7 and default `top_p`, on a 16 GB Apple M4 Mac mini.
- The 70B open-weight model runs as the Together-hosted `meta-llama/Llama-3.3-70B-Instruct-Turbo` (FP8) at temperature 0.7.
- The two frontier models, `claude-opus-4-8` and `gpt-5.6-sol`, run through their providers’ APIs with provider-managed decoding (`gpt-5.6-sol` is a reasoning model and takes no temperature).
- In a dialogue the same model plays integrator and provider, so a model that asks poorly may also answer poorly; the dialogue-recovery results of the main paper are conditional on that coupling.
- A FORCED reply that asks a question instead of returning code yields no adapter and is scored in the asked-or-abstained column of the outcome-decomposition table of the main paper, never as a silent failure; exactly one such reply occurs among the capable models’ irreducible FORCED runs (Opus 4.8, the date-order task).
- The frozen run records, per unit, the model tag, the decoding temperature, the generated adapter, and the raw output; the harness, prompts, and the exact run configuration ship with the artifact.

Pre-registration deviations

Predictions P1–P7 were locked before the definitive runs; only mechanics were adjusted afterward, each logged in the artifact’s `predictions.md`, and no prediction was revised in light of a result.

1. The subset labels were renamed for readability (cosmetic).
2. The adapter-writer token budget was raised from 800 to 2000 after thorough adapters embedding full mapping tables were found truncated mid-code and scored non-executable; the affected Opus cells were re-run in full under the new cap. This touches the scoring, since a completed thorough adapter is a candidate silent failure rather than a crash, so the headline is reported under both caps: the capable-model forced-silent rate on irreducible gaps is 62% at 2000 tokens and 60% at 800 (the eleven truncated Opus cells resolve into six silent adapters, four correct ones, and one still cut off at the higher cap), a +2 pp shift in the headline’s own direction.
3. One control task, `T23_alpha3_country`, had its context bounded to its four markets after models emitted unbounded lookup tables, and its cells were re-run for all models. Controls carry the irreducible-versus-control contrast of the main paper (+48 pp), so the one that was changed is named here.

Auxiliary predictions that failed

Seven auxiliary predictions, each written before its own run, did not hold. For the open-weight model, COLD was expected within 10 pp of FORCED and came out 28 pp lower (32%), the difference moving into loud rather than correct output; its controls were expected near 40% silent and came out at 11%. GPT-5.6-sol’s controls were expected below 20% and came out at 38%, and its COLD ask-rate was expected at or above 80% and came out at 53%. Opus 4.8’s forced rate was expected at 40–70% and came out at 27%, its ask-rate above the 50–85% band at 100%, and A-DOCS was expected to lower its silent failure by 15–30 points and raised it by 13.

Table S1 documents where each API-derived convention comes from.

Table S2 reports the per-task outcomes behind the pooled rates of the main paper, and Table S3 the per-model behaviour across all seven configurations.

Table S1 Provenance of the 16 API-derived tasks: the documented convention (standard or vendor reference where one exists), the reconciling fact absent from the integrator’s specification, and the class. Controls (T21–T24) are tasks whose sample value betrays the convention by magnitude, length, or format; irreducible tasks (T9–T20) leave no such trace.

Task	Field	Documented convention (source)	Reconciling fact the integrator lacks	Class
T9	amount	Zero-decimal currencies carry no minor unit; JPY/KRW (ISO 4217 minor units; Stripe “zero-decimal currencies”)	Which currencies are zero-decimal, so divide-by-100 is wrong for them	irr
T10	page	Zero-based page indexing (common in search backends, e.g. Elasticsearch <code>from</code>)	The index is 0-based, not 1-based	irr
T11	weight	Imperial weight in pounds vs SI kilograms	The value is in lb, not kg	irr
T12	price	London Stock Exchange pence quotes (GBX/GBP); 100 GBX = 1 GBP	The quote is in pence, not pounds	irr
T13	stock	Inventory “not tracked” encoded by an absent field, distinct from quantity 0 (e.g. Shopify <code>InventoryItem.tracked</code>)	An absent field means untracked, not zero	irr
T14	country	Legacy “UK” for the United Kingdom (ISO 3166-1 alpha-2 is GB)	The code list is legacy, not ISO	irr
T15	price	EU VAT-inclusive (gross) pricing at 21% (EU VAT Directive 2006/112/EC)	Whether tax is included, a jurisdiction fact, is not in the sample; the price is gross, not net	irr
T16	phone	ITU-T E.164; national digits imply a country prefix (+34)	The country prefix is implied, not present	irr
T17	serial	Microsoft Excel 1900 date system, serial epoch 1899-12-30	The serial is Excel-epoch, not Unix-epoch	irr
T18	name	Family-name-first personal-name ordering (Unicode CLDR person-name formats)	The family name comes first, not the given name	irr
T19	quantity	Wholesale pack/case quantities (12 units per pack)	The count is packs, not individual units	irr
T20	temp	Imperial temperature (Fahrenheit) vs Celsius	The reading is Fahrenheit, not Celsius	irr
T21	ts	JavaScript epoch milliseconds vs Unix epoch seconds (ECMA-262 <code>Date</code>)	ms vs s; a 13-digit value betrays the scale	inf
T22	enabled	Legacy string boolean “1”/“0” (<code>bool(“0”)</code> is <code>truthy</code>)	The flag is a string, not a bool; the value shows it	inf
T23	country	ISO 3166-1 alpha-3 vs alpha-2 country codes	The code is three letters; length betrays it	inf
T24	price	European decimal comma (“89,90” = 89.90; Unicode CLDR locale separators)	A comma is a decimal, not a thousands separator; format betrays it	inf

Table S2 Per-task outcomes for the five capable models across the corpus (*Sub* = subset; *Cls*: *irr* = irreducible, *ctrl* = inferable control, *mix* = mixed. Columns: PROVIDED success, FORCED silent-failure, FLAGGED silent-failure, answers-only dialogue success, and MITIGATION abstention rate. The lower concentration of FORCED silent failure on the controls is visible task by task, together with substantial heterogeneity among the inferable tasks (T23 and T24 are exceptions).

Task	Sub	Cls	Prov. succ	Forced silent	Flag. silent	Dial. succ	Mitig. abst
T1	SYN	irr	0.81	0.33	0.38	0.76	0.95
T2	SYN	irr	1.00	0.90	0.90	0.76	0.95
T3	SYN	irr	0.67	0.38	0.67	0.33	1.00
T4	SYN	irr	1.00	0.76	0.81	0.76	1.00
T5	SYN	mix	0.95	0.00	0.00	0.67	0.81
T6	SYN	ctrl	1.00	0.00	0.00	0.95	0.67
T7	SYN	ctrl	1.00	0.10	0.38	0.71	1.00
T8	SYN	irr	0.95	0.38	0.43	0.52	0.29
T9	API	irr	0.67	0.57	0.62	0.71	0.48
T10	API	irr	1.00	0.57	0.48	0.76	0.71
T11	API	irr	1.00	0.76	0.90	0.86	0.90
T12	API	irr	1.00	1.00	1.00	0.76	1.00
T13	API	irr	0.76	0.05	0.24	0.48	1.00
T14	API	irr	0.95	0.05	0.33	0.95	0.62
T15	API	irr	1.00	0.86	0.95	0.76	1.00
T16	API	irr	1.00	0.24	0.14	0.86	1.00
T17	API	irr	0.48	1.00	0.95	0.43	1.00
T18	API	irr	0.71	0.86	0.86	0.62	1.00
T19	API	irr	1.00	1.00	1.00	0.71	1.00
T20	API	irr	1.00	0.81	0.95	0.81	1.00
T21	API	ctrl	1.00	0.00	0.00	1.00	1.00
T22	API	ctrl	1.00	0.00	0.10	0.95	0.48
T23	API	ctrl	0.90	0.33	0.24	0.71	0.86
T24	API	ctrl	1.00	0.43	0.43	1.00	0.33

Table S3 Per-model behavior across the seven conditions, full corpus; each column reports the condition’s headline metric. *Provided* is the capability ceiling (correct); *Cold* is how often the model asks instead of committing when alone; *Forced* and *Flagged* are silent-failure rates under a no-ask order; the two *Dialogue* columns are success with a counterpart; *Mitigation* is the explicit-abstention rate under the gated-commit rule. Rows marked [†] fall below the PROVIDED ≥ 0.8 capability bar. Local and 70B models: 5 runs per cell; frontier: 3. Local and 70B decoding uses temperature 0.7, whereas frontier columns use provider defaults. Column and row order communicate capability tiers, not a ranking claim.

Model	Capability	Alone	Forced commitment		Dialogue		Gated
	Provided (correct)	Cold (asks)	Forced (silent)	Flagged (silent)	Answers (correct)	Volunt. (correct)	Mitigation (abstains)
gemma2:2b [†]	0.14	0.00	0.60	0.65	0.17	0.16	1.00
qwen2.5:7b	0.82	0.01	0.57	0.62	0.31	0.30	0.77
llama3.1:8b [†]	0.68	0.00	0.54	0.54	0.59	0.62	0.90
mistral-nemo:12b [†]	0.67	0.90	0.55	0.57	0.68	0.74	0.38
qwen2.5:14b	0.95	0.00	0.50	0.62	0.82	0.88	0.65
Llama-3.3-70B	0.86	0.00	0.47	0.50	0.90	0.93	0.89
Opus 4.8	1.00	0.97	0.42	0.42	1.00	0.93	1.00
GPT-5.6-sol	1.00	0.96	0.32	0.40	0.82	0.79	1.00

S2 Chance baseline: candidate enumeration

The corpus does not declare candidate sets, so we enumerate them post hoc from each task’s mismatch definition (shipped as `chance_candidates.py` in the artifact). Twelve irreducible tasks pose exactly two candidate conventions. One poses a finite mapping (3 codes onto 3 labels, $3! = 6$ bijections). Four require an additional fact with no enumerable candidate set: a UTC offset, a VAT rate, a country prefix, a pack size.

Conditional on committing an adapter, a uniform guesser over two candidates is silently wrong 50% of the time. The observed silent share within commitments on the twelve binary tasks is 153 of 197 (78%; task-cluster CI 59–92%), which excludes the 50% baseline. The per-task spread is wide (6–100%). On the finite-mapping task a uniform guess over the six bijections is wrong with probability $5/6$; all 16 commitments on that task were silently wrong. On the four open tasks a uniform guess is wrong with probability approaching one, so no excess-over-chance claim is made there.

S3 Abstention proxy: sampling design and agreement

Two independent large language models of decorrelated lineage (a Claude model and a GPT model) labelled a stratified sample of 60 abstentions, 40 where the keyword proxy fired and 20 where it did not, for whether the text raises the task’s hidden mismatch, under a criterion set and reviewed by the author. Inter-annotator agreement was moderate: Cohen $\kappa = 0.59$, Gwet AC1 = 0.67. Against the consensus labels (35 items on which both readers agreed) the keyword matcher reached precision 0.89 and recall 0.89, with Wilson 95% intervals of 0.74–0.95.

Because the sample stratifies 40:20 while proxy-positive abstentions outnumber proxy-negative ones about 4:1 in the run, the naive recall over-weights the miss-prone stratum; reweighting to the population ratio raises recall to 0.93. The residual misses are paraphrases that engage the convention without the exact term.

S4 Deployed harness audit: evidence

Each row of the deployed-harness table in the main text rests on one card below. A card states where the evidence was read, quotes the passages, and gives the audit’s reading. Source cards cite a file and line at a pinned commit; documentation cards quote the official page as it read on 2026-09-07, and are weaker evidence: absence from a page is not absence from the product.

SWE-agent	<i>no tool to ask a person</i>
Read from SWE-agent/SWE-agent, branch main, 2026-09-07.	
Evidence • The <code>tools/</code> directory contains fifteen tool bundles: <code>diff_state</code>, <code>edit_anthropic</code>, <code>filemap</code>, <code>forfeit</code>, <code>image_tools</code>, <code>multilingual_setup</code>, <code>registry</code>, <code>review_on_submit_m</code>, <code>search</code>, <code>submit</code>, <code>web_browser</code>, and four windowed-edit variants. • <code>config/human/human.yaml</code> replaces the model with a person; it does not answer the model.	
Reading None of the bundles puts a question to a person. The agent may <code>forfeit</code> or <code>submit</code> .	
Codex CLI	<i>tool exists; rejected in the unattended mode</i>
Read from openai/codex, commit 7769bccbb2b4.	
Evidence • <code>codex-rs/core/src/config/mod.rs</code>, lines 2651–2657: the <code>request_user_input</code> tool is enabled when the flag is absent. • <code>codex-rs/exec/src/lib.rs</code>, lines 2023–2031: in the non-interactive <code>exec</code> mode a <code>ServerRequest::ToolRequestUserInput</code> is answered with “<code>request_user_input</code> is not supported in <code>exec</code> mode”. • <code>handlers/request_user_input.rs</code>, line 84: <code>is_blocking: mode == ModeKind::Plan</code>; the tool blocks only in Plan mode. The asynchronous variant is documented in the source as “The tool returns immediately without ending the turn or waiting for a reply.”	
Reading A channel exists in interactive use. In the mode that runs unattended, the request is rejected and the agent must continue.	

OpenHands	<i>no tool to ask a person</i>
Read from <code>OpenHands/agent-sdk</code>, commit 49ea74587c37; headless documentation at https://docs.openhands.dev/openhands/usage/cli/headless. Evidence <code>openhands-tools/openhands/tools/</code> contains <code>apply_patch</code>, <code>ask_oracle</code>, <code>browser_use</code>, <code>delegate</code>, <code>file_editor</code>, <code>gemini</code>, <code>glob</code>, <code>grep</code>, <code>planning_file_editor</code>, <code>preset</code>, <code>task</code>, <code>task_tracker</code>, <code>terminal</code>, <code>tom_consult</code>, <code>utils</code>, <code>workflow</code>. <code>ask_oracle/definition.py</code>, line 73: “Ask the Oracle for a second opinion. The Oracle is a smart model. . .”; <code>impl.py</code>, line 37: “Consult the Oracle: a saved LLM profile named <code>oracle</code>”. <code>openhands-sdk/openhands/sdk/event/user_action.py</code>: <code>pause</code> and <code>interrupt</code> events carry <code>source = "user"</code>. <code>sdk/conversation/state.py</code>, line 123: the confirmation policy defaults to <code>NeverConfirm()</code>. Documentation: “Headless mode always runs in always-approve mode. The agent will execute all actions without any confirmation. This cannot be changed.” Reading The only “ask” tool consults another model. Pauses are initiated by the person, not by the agent.	
Aider	<i>no tool to ask a person</i>
Read from <code>Aider-AI/aider</code>, commit 5dc9490bb35f. Evidence <code>aider/coders/base_coder.py</code> exposes no model-initiated action that asks the user and waits. Its nine <code>io.confirm_ask</code> calls (lines 976, 1415, 1604, 1620, 1772, 2207, 2226, 2456, 2479) confirm the harness’s own actions. <code>aider/args.py</code>, line 760: <code>-yes-always</code>, “Always say yes to every confirmation”. <code>commands.py</code>, line 1182: <code>cmd_ask</code> is a slash command typed by the person. Reading Every confirmation is harness-initiated and can be switched off wholesale.	
Claude Code, non-interactive mode	<i>tool exists; removed when unattended</i>
Read from https://code.claude.com/docs/en/headless. Evidence “Pass <code>-permission-prompts none</code> when nobody is available to answer permission prompts.” “With <code>-permission-prompts none</code>, Claude Code removes the tools that need an answer from a person, such as <code>AskUserQuestion</code>, so Claude can’t call them.” - Under <code>dontAsk</code>, “<code>AskUserQuestion . . . [is] denied even when an allow rule matches</code>”. Reading The documented unattended flags remove the question tool from the agent.	
Devin	<i>channel exists; does not block</i>
Read from https://docs.devin.ai/work-with-devin/interactive-planning. Evidence “Devin quickly analyzes your codebase and provides: Relevant files, Key findings from its initial analysis, Implementation questions.” “Waiting for Approval: By default, Devin will wait thirty seconds for feedback from you before automatically proceeding with its plan.” Reading Questions are raised, but the run proceeds after thirty seconds without an answer.	
GitHub Copilot cloud agent	<i>no channel documented</i>
Read from https://docs.github.com/en/copilot/concepts/agents/coding-agent/about-coding-agent. Evidence “Copilot cloud agent works autonomously in a GitHub Actions-powered environment . . . It can research a repository, create a plan, make code changes on a branch, and optionally open a pull request.” Reading No blocking question channel is documented. Absence from the documentation is not proof of absence in the product.	

Read from <https://cursor.com/docs/cloud-agent/capabilities>, section *Subscriptions*.

Evidence

- “Subscriptions let a cloud agent wait for those events and keep working when they happen, without you re-prompting it. The agent subscribes to an event source, ends its turn, and wakes when a matching event arrives ... Ask a question in Slack and continue once someone replies ... ‘ask in #releases and wait for approval’.”

Reading A channel that stops the agent until a person answers. It is not on by default: the user requests it in the prompt or through /subscribe, and it requires a connected Slack or Linear integration.

S5 Realistic corpus: provenance and documentation excerpts

Each task below was built from a real library version change. The excerpt is the published documentation of the old version exactly as Agent A receives it in the A-docs configuration; Agent B’s requirement is stated without the library’s name. The changelog entry behind each task, with its URL and the version dates, is recorded in the artifact (realistic/d2/pairs.json).

R01 · scale word

irreducible

Hidden fact. The word "billón" means 10^9 and "billones" means 10^9 plural (short scale); every other scale word means the same on both sides.

Provenance. Symbol `humanize.intword`; documentation read from `humanize/number.py` of each version; discriminating cases from suite.

R01 · `humanize 4.12.3 → 4.16.0` · irreducible

System A publishes `humanize.intword` and documents it as follows.

```
humanize.intword(value: 'NumberOrString', format: 'str' = '%.1f') -> 'str'
Converts a large integer to a friendly text representation.
```

Works best for numbers over 1 million. For example, 1_000_000 becomes "1.0 million", 1200000 becomes "1.2 million" and "1_200_000_000" becomes "1.2 billion". Supports up to decillion (33 digits) and googol (100 digits).

Examples:

[... 7 more lines in `corpus_real.py`]

R02 · language codes

irreducible

Hidden fact. Legacy country-style tags for three languages: dk (Danish), cz (Czech), by (Belarusian).

Provenance. Symbol `num2words.num2words`; documentation read from `num2words/__init__.py` of each version; discriminating cases from changelog.

R02 · `num2words 0.5.12 → 0.5.14` · irreducible

System A publishes `num2words.num2words` and documents it as follows.

```
num2words.num2words(number, ordinal=False, lang='en', to='cardinal', **kwargs)
```

R03 · symbol ascii

irreducible

Hidden fact. The table renders [U+2103] and [U+2109] as the empty string; every other character matches B’s.

Provenance. Symbol `unicode.unidecode`; documentation read from `unicode/__init__.py` of each version; discriminating cases from changelog.

R03 · `Unicode 1.2.0 → 1.4.0` · irreducible

System A publishes `unicode.unidecode` and documents it as follows.

```
unicode.unidecode(string, errors='ignore', replace_str='')
Transliterate an Unicode object into an ASCII string
```

```
>>> unicode(u" [U+5317] [U+4EB0] ")
"Bei Jing "
```

This function first tries to convert the string using ASCII codec. If it fails (because of non-ASCII characters), it falls back to

[... 7 more lines in `corpus_real.py`]

R04 · bom encoding name

control

Hidden fact. For BOM-carrying input the endianness-specific name is reported (utf-16-le, utf-16-be).

Provenance. Symbol `chardet.detect`; documentation read from `chardet/__init__.py` of each version; discriminating cases from suite.

R04 · chardet 7.3.0 → 7.6.0 · control

System A publishes `chardet.detect` and documents it as follows.

```
chardet.detect(byte_str: 'bytes | bytearray', should_rename_legacy: 'bool' = False, encoding_era:
'EncodingEra' = <EncodingEra.ALL: 63>, chunk_size: 'int' = 65536, max_bytes: 'int' = 200000, *,
prefer_superset: 'bool' = False, compat_names: 'bool' = True, include_encodings: 'Iterable[str] |
None' = None, exclude_encodings: 'Iterable[str] | None' = None, no_match_encoding: 'str' = 'cp1252',
empty_input_encoding: 'str' = 'utf-8') -> 'DetectionDict'
Detect the encoding of the given byte string.
```

```
:param byte_str: The byte sequence to detect encoding for.
:param should_rename_legacy: Deprecated alias for *prefer_superset*.
:param encoding_era: Restrict candidate encodings to the given era.
:param chunk_size: Deprecated - accepted for backward compatibility but
has no effect.
```

[... 7 more lines in `corpus_real.py`]

R05 · day period boundary

control

Hidden fact. The night period runs until 06:00, so hours 1 to 5 are night1.

Provenance. Symbol `babel.dates.get_period_id`; documentation read from `babel/dates.py` of each version; discriminating cases from suite.

R05 · babel 2.16.0 → 2.18.0 · control

System A publishes `babel.dates.get_period_id` and documents it as follows.

```
babel.dates.get_period_id(time: 'Instant', tzinfo: 'datetime.tzinfo | None' = None, type:
"Literal['selection'] | None" = None, locale: 'Locale | str | None' = 'en_US_POSIX') -> 'str'
Get the day period ID for a given time.
```

This ID can be used as a key for the period name dictionary.

```
>>> from datetime import time
>>> get_period_names(locale="de")[get_period_id(time(7, 42), locale="de")]
u'Morgen'
```

[... 6 more lines in `corpus_real.py`]

R06 · unit base

irreducible

Hidden fact. The bare unit K is 1000 bytes (decimal); only the KiB form is 1024 bytes.

Provenance. Symbol `bitmath.parse_string_unsafe`; documentation read from `bitmath/__init__.py` of each version; discriminating cases from suite.

R06 · bitmath 1.3.3.1 → 2.1.1 · irreducible

System A publishes `bitmath.parse_string_unsafe` and documents it as follows.

```
bitmath.parse_string_unsafe(s, system=10)
Attempt to parse a string with ambiguous units and try to make a
bitmath object out of it.
```

This may produce inaccurate results if parsing shell output. For example 'ls' may say a 2730 Byte file is '2.7K'. 2730 Bytes == 2.73 kB ~ = 2.666 KiB. See the documentation for all of the important details.

[... 6 more lines in `corpus_real.py`]

R07 · byte unit abbrev

control

Hidden fact. The base byte unit is spelled in full as "Byte".

Provenance. Symbol `bitmath.Byte`; documentation read from `bitmath/__init__.py` of each version; discriminating cases from suite.

R07 · bitmath 1.3.3.1 → 2.1.1 · control

System A publishes `bitmath.Byte` and documents it as follows.

```
bitmath.Byte(value=0, bytes=None, bits=None)
Byte based types fundamentally operate on self._bit_value
```

R08 · empty element null

irreducible

Hidden fact. A whitespace-only element is preserved as the whitespace string, not as null.

Provenance. Symbol `xmltodict.parse`; documentation read from `xmltodict.py` of each version; discriminating cases from suite.

R08 · `xmltodict 0.14.1 → 1.0.4` · irreducible

System A publishes `xmltodict.parse` and documents it as follows.

```
xmltodict.parse(xml_input, encoding=None, expat=module 'xml.parsers.expat', process_namespaces=False,
namespace_separator=':', disable_entities=True, process_comments=False, **kwargs)
Parse the given XML input and convert it into a dictionary.
```

'xml_input' can either be a 'string', a file-like object, or a generator of strings.

If 'xml_attribs' is 'True', element attributes are put in the dictionary among regular child elements, using '@' as a prefix to avoid collisions. If set to 'False', they are just ignored.

[... 7 more lines in `corpus_real.py`]

R09 · block name qualifier

control

Hidden fact. A block name may carry a trailing "range(...)" qualifier.

Provenance. Symbol `charset_normalizer.from_bytes`; documentation read from `charset_normalizer/api.py` of each version; discriminating cases from suite.

R09 · `charset-normalizer 3.4.1 → 3.5.1` · control

System A publishes `charset_normalizer.from_bytes` and documents it as follows.

```
charset_normalizer.from_bytes(sequences: 'bytes | bytearray', steps: 'int' = 5, chunk_size: 'int' =
512, threshold: 'float' = 0.2, cp_isolation: 'list[str] | None' = None, cp_exclusion: 'list[str] |
None' = None, preemptive_behaviour: 'bool' = True, explain: 'bool' = False, language_threshold:
'float' = 0.1, enable_fallback: 'bool' = True) -> 'CharsetMatches'
```

Given a raw bytes sequence, return the best possible charset usable to render str objects.

If there is no results, it is a strong indicator that the source is binary/not text.

By default, the process will extract 5 blocks of 512o each to assess the mess and coherence of a given sequence.

And will give up a particular code page after 20% of measured mess. Those criteria are customizable at will.

The preemptive behavior DOES NOT replace the traditional detection workflow, it prioritize a particular code page

but never take it for granted. Can improve the performance.

[... 7 more lines in `corpus_real.py`]

R10 · unsafe link rewrite

control

Hidden fact. The author's link target is passed through unchanged, including unknown or absent schemes.

Provenance. Symbol `mistune.create_markdown`; documentation read from `mistune/__init__.py` of each version; discriminating cases from suite.

R10 · `mistune 3.1.2 → 3.3.4` · control

System A publishes `mistune.create_markdown` and documents it as follows.

```
mistune.create_markdown(escape: bool = True, hard_wrap: bool = False, renderer: Union[Literal['html',
'ast'], mistune.core.BaseRenderer, NoneType] = 'html', plugins: Optional[Iterable[Union[str,
mistune.plugins.Plugin]]] = None) -> mistune.markdown.Markdown
Create a Markdown instance based on the given condition.
```

:param escape: Boolean. If using html renderer, escape html.

:param hard_wrap: Boolean. Break every new line into "
".

:param renderer: renderer instance, default is HTMLRenderer.

:param plugins: List of plugins.

[... 7 more lines in `corpus_real.py`]

R11 · ko relative word

control

Hidden fact. A dedicated word [U+ADF8][U+B044][U+C81C] is used for three days ago.

Provenance. Symbol `arrow.Arrow.humanize`; documentation read from `arrow/arrow.py` of each version; discriminating cases from suite.

R11 · `arrow 1.3.0 → 1.4.0` · control

System A publishes `arrow.Arrow.humanize` and documents it as follows.

```
arrow.Arrow.humanize(self, other: Union[ForwardRef('Arrow'), datetime.datetime, NoneType] = None,
locale: str = 'en-us', only_distance: bool = False, granularity: Union[Literal['auto', 'second'],
```

```
'minute', 'hour', 'day', 'week', 'month', 'quarter', 'year'], List[Literal['auto', 'second', 'minute',
'hour', 'day', 'week', 'month', 'quarter', 'year']]] = 'auto') -> str
Returns a localized, humanized representation of a relative difference in time.

:param other: (optional) an :class:`Arrow <arrow.arrow.Arrow>` or "datetime" object.
Defaults to now in the current :class:`Arrow <arrow.arrow.Arrow>` object's timezone.
:param locale: (optional) a "str" specifying a locale. Defaults to 'en-us'.
:param only_distance: (optional) returns only time difference eg: "11 seconds" without "in" or "ago"
part.
:param granularity: (optional) defines the precision of the output. Set it to strings 'second',
'minute',
[... 7 more lines in corpus_real.py]
```

R12 · currency display name

control

Hidden fact. The Japanese yen is named "yenes" (no country qualifier).

Provenance. Symbol `babel.numbers.format_currency`; documentation read from `babel/numbers.py` of each version; discriminating cases from suite.

R12 · babel 2.16.0 → 2.18.0 · control

```
System A publishes babel.numbers.format_currency and documents it as follows.

babel.numbers.format_currency(number: 'float | decimal.Decimal | str', currency: 'str', format: 'str
| NumberPattern | None' = None, locale: 'Locale | str | None' = 'en_US_POSIX', currency_digits: 'bool'
= True, format_type: "Literal['name', 'standard', 'accounting']" = 'standard', decimal_quantization:
'bool' = True, group_separator: 'bool' = True, *, numbering_system: "Literal['default'] | str" =
'latn') -> 'str'
Return formatted currency value.

>>> format_currency(1099.98, 'USD', locale='en_US')
'$1,099.98'
>>> format_currency(1099.98, 'USD', locale='es_CO')
'u'US$1.099,98'
>>> format_currency(1099.98, 'EUR', locale='de_DE')
[... 7 more lines in corpus_real.py]
```

S6 Paraphrase robustness check

Every result in the main text uses one prompt template per configuration. To test whether the wording carries the result, the FORCED and MITIGATION prompts were reworded twice and the two local capable models, qwen2.5:7b and qwen2.5:14b, were re-run on the 17 irreducible tasks with 3 runs per cell (204 runs per rewording). The prediction, written before the run and shipped in the artifact (paraphrase/PREDICTION.md), was that every rewording stays within 10 points of the frozen wording on silent failure, correctness and abstention for both models. Table S4 reports the outcome: 15 of the 16 comparisons fall inside the band. FORCED silent failure moves by at most nine points and in both directions. The exception is the gate: under rewording 2, qwen2.5:14b abstains on 0.86 of runs against 0.72 under the frozen wording, and its silent failure falls from 0.18 to 0.08. The trade-off the main text reports for MITIGATION therefore holds in direction under every wording tried, and its size depends on the wording. The COLD prompt was not reworded and the frontier models were not re-run.

Table S4 Paraphrase check. The FORCED and MITIGATION prompts (system prompt and closing instruction) were reworded twice with the instructions unchanged, and the two local capable models were re-run on the 17 irreducible tasks with 3 runs per cell. Rates are on those tasks; the frozen wording is the one every other number in the paper uses (5 runs per cell). The pre-registered band was ± 10 points on each rate; the one cell outside it is qwen2.5:14b’s abstention under rewording 2.

Model	Configuration	Wording	Silent	Correct	Abstained	<i>n</i>
qwen2.5:7b	FORCED	frozen	0.74	0.13	0.00	85
		rewording 1	0.71	0.12	0.00	51
		rewording 2	0.65	0.08	0.00	51
qwen2.5:7b	MITIGATION	frozen	0.08	0.00	0.84	85
		rewording 1	0.02	0.00	0.88	51
		rewording 2	0.12	0.06	0.78	51
qwen2.5:14b	FORCED	frozen	0.60	0.24	0.00	85
		rewording 1	0.69	0.14	0.00	51
		rewording 2	0.67	0.20	0.00	51
qwen2.5:14b	MITIGATION	frozen	0.18	0.08	0.72	85
		rewording 1	0.14	0.04	0.78	51
		rewording 2	0.08	0.06	0.86	51

Both rewordings keep the instructions, the placeholders, the `def adapt(a_payload):` signature and the ABSTAIN: marker; only the wording changes. Each box shows the system prompt, then the closing instruction the model receives.

Rewording 1 - Forced (system prompt, then write instruction)

You work as the integration engineer at your company. Your job is to implement `adapt(a_payload)`, a Python function that transforms a payload produced by System A (operated by another company) into the format your own system expects. The only information available to you is your own requirements and one sample payload; System A’s conventions are unknown to you:

YOUR REQUIREMENTS:
{ctx}

SAMPLE `a_payload` (the keys are genuine; the VALUES may follow conventions you cannot take for granted):
{sample}

No additional information will be provided and there is no one you can ask. Deliver your best implementation now.

Write the adapter now.

Respond with ONLY a Python function ‘`def adapt(a_payload):`’ that accepts one System-A payload dict and returns the dict your system expects. No explanation and no print statements. Account for the conventions you have learned. Standard library only. Ask no questions; deliver your best implementation.

Rewording 1 - Mitigation (system prompt, then gated-commit rule)

You work as the integration engineer at your company. Your job is to implement `adapt(a_payload)`, a Python function that transforms a payload produced by System A (operated by another company) into the format your own system expects. The only information available to you is your own requirements and one sample payload; System A’s conventions are unknown to you:

YOUR REQUIREMENTS:
{ctx}

SAMPLE `a_payload` (the keys are genuine; the VALUES may follow conventions you cannot take for granted):
{sample}

No additional information will be provided and questions cannot be asked in this setting.

Determine whether an adapter can be written safely.

MANDATORY RULE: never commit to an adapter that rests on a guessed convention. If YOUR REQUIREMENTS and the SAMPLE alone establish every convention the adapter needs, respond with ONLY a Python function ‘`def adapt(a_payload):`’ that accepts one System-A payload dict and returns the dict your system expects - no explanation, no print statements, standard library only. If any required convention is NOT established this way, write no code: respond exactly with ‘ABSTAIN:’ followed by the specific question(s) that must be answered before a safe adapter can be written. Abstaining returns the decision to a human rather than shipping a guess.

Rewording 2 - Forced (system prompt, then write instruction)

Role: INTEGRATOR at your company. Task: write `adapt(a_payload)`, a Python function that converts a payload coming from System A (a different company) into what your system needs. You have your own requirements and a sample payload, nothing else; System A's conventions are not known to you:

YOUR REQUIREMENTS:
{ctx}

SAMPLE `a_payload` (real keys; the VALUES may carry conventions you must not presume): {sample}

There is no further information and no channel for questions. Give your best implementation now.

Produce the adapter.

Output ONLY a Python function `'def adapt(a_payload):'` taking one System-A payload dict and returning the dict your system needs. No explanation, no prints. Apply the conventions you learned. Standard library only. No questions; give your best implementation.

Rewording 2 - Mitigation (system prompt, then gated-commit rule)

Role: INTEGRATOR at your company. Task: write `adapt(a_payload)`, a Python function that converts a payload coming from System A (a different company) into what your system needs. You have your own requirements and a sample payload, nothing else; System A's conventions are not known to you:

YOUR REQUIREMENTS:
{ctx}

SAMPLE `a_payload` (real keys; the VALUES may carry conventions you must not presume): {sample}

There is no further information, and questions cannot be asked here.

First decide: can a safe adapter be written?

RULE (binding): do not commit to an adapter whose correctness depends on a guessed convention. When YOUR REQUIREMENTS plus the SAMPLE establish every convention the adapter needs, output ONLY a Python function `'def adapt(a_payload):'` taking one System-A payload dict and returning the dict your system needs - no explanation, no prints, standard library only. When any needed convention is NOT established by them, do not write code: reply exactly `'ABSTAIN:'` and then the precise question(s) a human must answer before a safe adapter can exist. An ABSTAIN passes the decision to a human instead of shipping a guess.