

Code Agents under Split Specifications: An Empirical Study of Silent Failure in System Integration

Camilo Chacón Sartori

`camilo.chacon@apeironagents.tech`

Apeiron Intelligence

Research Article

Keywords: Large language model (LLM) code agents, system integration, interoperability, silent failure, multi-agent systems, empirical software engineering

Posted Date: October 13th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11027985/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Code Agents under Split Specifications: An Empirical Study of Silent Failure in System Integration

Camilo Chacón Sartori^{1*}

^{1*}Apeiron Intelligence, Spain.

Corresponding author(s). E-mail(s): camilo.chacon@apeironagents.tech;

Abstract

Integrations across organizational boundaries depend on facts that only one side records: a date order, a currency unit, a legacy code. When a code agent driven by a large language model (LLM) writes the adapter without that fact, it does not crash. It picks a convention: the adapter passes the ordinary cases and returns a wrong value on the one that needed it, a silent failure. We measure this on two corpora, scoring each adapter by executing it against tests. On 24 authored tasks, the five models able to build these adapters fail silently on 62% of runs over the 17 tasks whose fact cannot be inferred, when the pipeline forbids the question. The cost depends on the pipeline, not the model: the two frontier systems never commit silently when they may ask, yet fail silently on 59% and 45% when they may not. An audit of eight deployed agent harnesses finds six running by default with no channel that blocks on the agent’s question. On 12 further tasks built from real library version changes, the three capable models fail silently on between 27% and 60% of runs; reading the counterpart’s documentation helps two and hurts one. A counterpart that answers what is asked lifts correctness from 9% to 70%; an abstention rule removes silent failure but cuts completion from 81% to 18%. The safeguard belongs in the pipeline: route the agent’s question to a person, and test on cases that discriminate between conventions.

Keywords: Large language model (LLM) code agents, system integration, interoperability, silent failure, multi-agent systems, empirical software engineering

1 Introduction

Integration across organizational boundaries depends on facts that only one side records, and this paper measures what an LLM code agent does when it must build the adapter without them. The underlying problem is a classic of software engineering: components built by different teams rarely agree on data models, encodings, or conventions, and reconciling those mismatches has driven decades of work on heterogeneous data integration and schema matching [1, 2]. Integration across organizational boundaries is that problem under split information. System A exposes an interface; System B consumes it; the correctness of the join depends on facts recorded on the far side: whether a date is day-first or month-first, whether an amount is in cents or euros, what a legacy status code means. These facts are not in the payload, and a sample does not reveal them.

Code agents are increasingly handed exactly this work. When the reconciling fact is absent from the agent’s own specification, the failure that concerns us is not a crash. The agent commits to a convention it was never given. The adapter compiles, passes the common cases, and returns a wrong value on the case that needed the fact. We call this outcome a *silent failure*. It is a costly shape of integration error precisely because nothing raises: the code review shows a plausible adapter, and the wrong values flow downstream until an external check detects them. Figure 1 shows the setting.

The central finding of this paper is that silent failure is a property of the pipeline, not of the model: the same two frontier systems commit no silent failure when the pipeline lets them ask, and commit silently on 59% and 45% of runs when it forbids the question. When the pipeline lets the agent ask, Claude Opus 4.8 and GPT-5.6-sol ask on 96–98% of runs on the tasks whose fact cannot be inferred. Their capability is identical in both settings: each builds the adapter on every run when handed the fact. What changed is the wiring. Section 3 describes the instrument that measures this.

An open channel is necessary but not sufficient, and asking does not by itself deliver. Three open-weight models that build these adapters reliably when given the fact almost never ask even when they may, and commit silently on a third to two thirds of runs. And when a frontier system does ask but the question reaches no one, the integration stalls: correctness across the five capable models falls to 9%. The exposure therefore splits by tier. Agents that ask leave a stalled integration; agents that do not ask ship a wrong value downstream.

What lowers the failure is moving the fact across the boundary before the agent commits. A counterpart that answers only what it is asked raises correctness from 9% to 70%. Naming the ambiguous field while still requiring a commitment does not help; it makes things slightly worse (Section 4.3). A rule that forbids commitment until every convention is established removes the silent failure but stalls most of the tasks whose convention was in fact recoverable from the interface (Section 4.7). The engineering target these results define is a pipeline that detects when a fact lies across a boundary and moves it before the agent commits.

Two terms recur throughout. We call a model *capable* on this corpus when it builds the adapter on at least 80% of runs given the fact; five of the eight models qualify. We call a task *irreducible* when the missing fact cannot be inferred from the integrator’s own specification and sample; 17 of the 24 tasks are irreducible, and the other six are

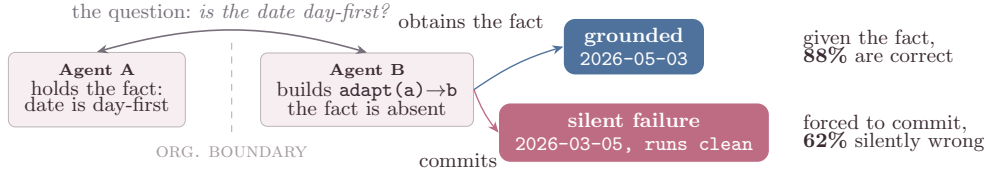


Fig. 1 The setting. Agent A’s specification records that dates are day-first; Agent B’s omits that fact although its adapter needs it. Obtaining the fact yields a correct adapter (upper path, **blue**); committing without it yields a silent failure (lower path, **rose**): the code runs without error and returns the wrong value.

controls where the interface does reveal the convention. Unless stated otherwise, every rate in this paper is for capable models on irreducible tasks.

Concretely, the paper establishes six things. First, capable agents commit silently on most runs when the pipeline forbids the question, and the rate does not fall when the ambiguous field is named for them: the cause is absent information, not missed detection or weak coding (Sections 4.2–4.3). Second, the tendency to ask does not track model scale (Section 4.5). Third, the two frontier systems leave different traces when they fail, one documenting its guess and the other leaving none, so a review check calibrated on one does not transfer (Section 4.4.1).

Fourth, dialogue recovers most of the capability ceiling, and the mechanism is the integrator’s own question rather than what the counterpart volunteers (Section 4.6). Fifth, a naive abstention gate trades silent failure for stalled completion (Section 4.7). Sixth, the same pattern appears on tasks whose mismatch, documentation, and tests come from real library version changes, and reading the counterpart’s documentation helps two of the three capable models and hurts the third (Section 4.8). The instrument behind these findings, two corpora of 24 and 12 tasks, seven pipeline configurations on the first and an eighth on the second, a deterministic execution oracle, and predictions registered before the definitive runs, is released as an artifact that regenerates every reported number without model access.

2 Background and related work

Interoperability as a software engineering problem

Reconciling heterogeneous schemas and conventions predates LLMs by decades: federated database systems [1] and automatic schema matching [2] treat the mismatch between independently designed systems as the central obstacle, and the discipline’s classic answer is to make the interface carry the load: information hiding confines design decisions behind module boundaries [3], and design by contract makes the mutual obligations explicit [4]. Software merging [5] and API evolution [6] study what breaks when independently developed artifacts meet; Dig and Johnson report that over 80% of breaking API changes are refactorings, structural changes of exactly the convention-shaped kind our tasks contain. Modern agent interoperability protocols standardize the connection layer [7], but a protocol standardizes how agents talk, not whether the reconciling fact crosses the boundary. Our setting is the case these

traditions assume away: a fact the interface does not carry and the sample cannot reveal.

Clarification behavior of code models

A line of work asks whether a single agent, given an underspecified requirement, poses a clarifying question, and reports that models frequently do not [8–12]. That setting is single-party: one agent clarifies a task with a user who holds the answer. Ours is bilateral: the resolving fact is held by another agent behind an organizational boundary, and either side could surface it.

Multi-agent behavior under distributed information

Studies of LLM collaboration under information asymmetry and partial observability examine whether agents exchange messages, produce decisions, and repair outputs effectively [13–15]. Closest to our result, a study of collective *reasoning* under distributed information [16] finds that splitting information collapses accuracy and that neither scale nor individual skill predicts the collapse. Where that result shows a wrong decision, ours shows an executable artifact that runs and returns the wrong value, scored by a deterministic oracle, with a capability control and a comparison that names the ambiguous field while withholding its convention, which together localize the cause in absent information.

Code agents in software engineering

Code agents are moving from single-function benchmarks [17, 18] to repository-level work: SWE-bench evaluates agents on real GitHub issues [19], a growing survey literature maps agentic SE systems [20], and multi-agent frameworks such as MetaGPT [21] and ChatDev [22] orchestrate role-structured pipelines that hand integration-shaped work to models. Empirical evaluations of LLM-based automation now span code-generation baselines for a new platform [23], vulnerability repair [24], game regression testing [25], and autonomous mobile automation [26]. These systems measure end-to-end task success; none isolates what happens at the knowledge boundary when the fact an integration needs sits in another organization. That boundary is our unit of analysis.

Reliability of generated code

Hallucination taxonomies for generated code catalogue outputs that deviate from intent or from real-world knowledge [27]. Recent agent evaluation uses *silent failure* for a different construct: an agent that asserts task completion while the environment state shows otherwise [28]. The silent failure measured here is artifact-level: the adapter itself executes cleanly and returns a wrong value, decided by the oracle regardless of what the agent asserts. EvalPlus shows that benchmark test suites systematically overstate correctness: adding discriminating tests drops measured pass rates [29], which is the same observation that motivates our benign/discriminating oracle split. Whether an LLM agent executes a natural-language test case reliably has likewise been treated as a quality question in its own right [30]; the silent failure measured here is the analogous question for the adapter the agent writes. The *acknowledged guess* shape we

report (Section 4.4.1) has a human mirror in self-admitted technical debt, where developers document a known shortcut and ship it [31]; in our data the annotation and the commitment co-occur in the same generated artifact.

Abstention and calibration

Language-model confidence can be calibrated against answer correctness in question-answering settings [32], and selective prediction trades coverage for accuracy by abstaining under uncertainty [33]. Our gated-commit rule (the agent writes the adapter only once every convention it needs is established, and otherwise abstains and names the missing question; Section 3.3) is selective prediction applied to integration commitment; the over-abstention we measure on the control tasks shows that this calibration does not transfer for free to the commit-or-abstain decision an integrating agent faces.

Coordination under partial knowledge in code generation

The specification gap [34] studies two code agents co-developing one class from an incomplete *shared* specification, where restoring the specification recovers coordination. Here the specification is instead *split across organizations*: the reconciling fact is recorded on the other side, absent from the integrating agent’s local specification, and must be obtained across the boundary rather than inferred locally. The error profile of AI-generated code also differs in origin from human coding error [35]; the silent failure we measure is an instance of that structural difference at the integration boundary.

3 Study design

We first describe the instrument (corpus, pipeline configurations, models, oracle, and capability gates) and then state the research questions in its terms (Section 3.7). The Supplementary Material (Online Resource 1) summarizes the protocol end to end in its Figure S1; every figure and table numbered with an S below is in that file.

3.1 Corpus

Each task defines System A’s output, System B’s required input, and at least one hidden mismatch that neither side’s specification resolves alone. Agent A receives only A’s context; Agent B receives only B’s context and one sample payload; together they must produce an adapter `adapt(a_payload) → b_payload`. The corpus has two subsets. *SYN* (8 tasks) isolates one convention family per task: date order, unit scale, timezone, a non-canonical enum map, a null-means-zero convention, a boolean encoding, an identifier shape, and list-versus-single cardinality. *API* (16 tasks) reproduces documented conventions of real interfaces and industry practice, some standardized and some widespread: zero-decimal currency amounts, zero-based pagination, pound-denominated weights, GBX (pence sterling) quotes, absent-means-untracked inventory, legacy country codes, VAT-inclusive (value-added tax) prices, national telephone digits, spreadsheet date serials, family-name-first person names, pack-counted quantities, Fahrenheit telemetry, and four inferable controls (epoch milliseconds, string flags, alpha-3 country codes, decimal commas).

Table 1 The 24 bilateral integration tasks. *SYN* tasks isolate one convention family each; *API* tasks reproduce documented interface and industry conventions. A task is *irreducible* when the hidden fact is not recoverable from the integrator’s context and the sample payload, and *inferable* (control) when the interface reveals it. Oracle cases are total / discriminating.

Task	Subset	Hidden fact	Domain	Class	Cases
T1	SYN	date order	e-commerce order sync	irreducible	4 / 3
T2	SYN	unit scale (cents)	payments reconciliation	irreducible	3 / 2
T3	SYN	timezone	event log federation	irreducible	2 / 2
T4	SYN	legacy enum map	CRM status mapping	irreducible	3 / 3
T5	SYN	null default + scale	pricing integration	mixed	3 / 2
T6	SYN	boolean Y/N	subscription flag sync	inferable	2 / 1
T7	SYN	id shape	customer master-data merge	inferable	2 / 2
T8	SYN	cardinality	catalog item feed	irreducible	2 / 1
T9	API	zero-decimal currency	payment-gateway settlement	irreducible	3 / 2
T10	API	pagination base	search-index pagination bridge	irreducible	2 / 2
T11	API	weight unit (lb)	cross-border shipping	irreducible	3 / 2
T12	API	GBX pence quote	equity market data	irreducible	3 / 2
T13	API	absent = untracked	inventory federation	irreducible	3 / 1
T14	API	legacy country code	customer-data onboarding	irreducible	3 / 1
T15	API	VAT-inclusive price	retail-to-ledger pricing	irreducible	3 / 2
T16	API	phone country prefix	CRM telephony sync	irreducible	2 / 2
T17	API	spreadsheet date serial	spreadsheet export ingestion	irreducible	2 / 2
T18	API	name order	HR directory import	irreducible	3 / 2
T19	API	pack quantity	wholesale order relay	irreducible	3 / 2
T20	API	temperature scale	industrial IoT telemetry	irreducible	3 / 2
T21	API	epoch milliseconds	telemetry timestamp bridge	inferable	2 / 2
T22	API	flag "1"/"0"	feature-flag sync	inferable	2 / 1
T23	API	alpha-3 country	trade-document routing	inferable	2 / 2
T24	API	decimal comma	invoice ingestion	inferable	2 / 2

Tasks are classified by whether the mismatch is *irreducible* (the fact is not recoverable from B’s context and the sample: 17 tasks) or *inferable* (the interface reveals the convention: 6 control tasks; one task mixes both). The corpus file is its own well-formedness proof: for every task, a reference adapter passes every oracle case, and a naive adapter written under B’s own convention passes every benign case and fails at least one discriminating case. Table 1 lists all 24 tasks.

3.2 Realistic corpus

A second corpus of 12 tasks draws its mismatches from real changes in ten published Python libraries rather than from the authors. When a library changes a convention between two published versions, code that produced values under the old version and code that consumes them under the new one are two systems separated by exactly the kind of fact this paper studies. That fact is recorded in a changelog which neither side’s documentation carries. We mined 49 libraries for such changes. We kept the 43 version pairs in which at least 80% of the old version’s test suite still runs against the new code, so that the change is one of convention and not of removed interface. Of those, 23 pairs show at least one silent divergence, an input on which the two versions return different values without raising; the others change interfaces or error behavior, which the oracle would score as loud. From the 23 we built 13 tasks.

A task enters the corpus only if an open-weight model (`qwen2.5:14b`) builds a correct adapter on at least four of five runs when handed the fact; this rule was applied before any pipeline configuration was run. It keeps the corpus to tasks a capable agent can solve from the fact and the standard library, and it declares the selection. Twelve of the 13 tasks, from ten libraries, passed it: five irreducible and seven controls. Their oracle carries 66 cases.

Each task that passes follows the schema of Section 3.1. System A emits values under the old convention, System B requires them under the new one, and the agent writes `adapt`; the hidden fact is the convention change. Agent A’s context is the published documentation of the old version. Agent B’s context is a plain statement of its requirement, without the library’s name. Discriminating oracle cases are inputs on which the two versions disagree, taken from the maintainers’ tests or from changelog-driven probes; benign cases are inputs on which the two versions agree, measured rather than assumed. Table 2 lists the tasks; the Supplementary Material gives every documentation excerpt with its provenance.

Table 2 The realistic corpus: tasks built from real library version changes. Each task’s hidden fact is the convention that changed between the two versions; *irreducible* and *control* follow Table 1. Oracle cases are total / discriminating.

Task	Library	Versions	Convention	Class	Cases
R01	humanize	4.12.3 → 4.16.0	scale word	irreducible	6 / 2
R02	num2words	0.5.12 → 0.5.14	language codes	irreducible	7 / 3
R03	Unidecode	1.2.0 → 1.4.0	symbol ASCII	irreducible	7 / 2
R04	chardet	7.3.0 → 7.6.0	BOM (byte-order mark) encoding name	control	5 / 1
R05	babel	2.16.0 → 2.18.0	day period boundary	control	10 / 2
R06	bitmath	1.3.3.1 → 2.1.1	unit base	irreducible	5 / 3
R07	bitmath	1.3.3.1 → 2.1.1	byte unit abbrev	control	4 / 1
R08	xmldict	0.14.1 → 1.0.4	empty element null	irreducible	4 / 2
R09	charset-normalizer	3.4.1 → 3.5.1	block name qualifier	control	4 / 1
R10	mistune	3.1.2 → 3.3.4	unsafe link rewrite	control	4 / 2
R11	arrow	1.3.0 → 1.4.0	Korean relative word	control	4 / 1
R12	babel	2.16.0 → 2.18.0	currency display name	control	6 / 1

The realistic corpus adds an eighth pipeline configuration, A-DOCS: as FORCED, but Agent B also receives the old version’s published documentation. It asks whether reading the counterpart’s documentation surfaces the change, which is the situation of an integrator who can read the other side’s repository but does not read its changelog.

3.3 Pipeline configurations

Each (task, model) pair runs under seven pipeline configurations. They differ in one thing only: whether, and how, the missing fact can cross the boundary before the agent commits.

Table 3 Deployed code-agent harnesses, audited 2026-09-07 in their default or unattended configuration. *Source* rows are read from the harness’s tool inventory at a pinned commit; *docs* rows are read from official documentation and are weaker evidence.

Harness	Tool to ask a person	Blocks by default	Evidence
SWE-agent	none (15 tools; forfeit or submit)	—	source
Codex CLI	exists; rejected in non-interactive exec mode	—	source
OpenHands	none to a person (ask_oracle consults a model)	—	source
Aider	none (harness confirmations; --yes-always)	—	source
Claude Code -p	exists; removed under the documented unattended flags	—	docs
Devin	exists	no: proceeds after 30 s	docs
Copilot cloud agent	not documented	—	docs
Cursor cloud agents	exists (subscriptions), on request	yes: ends its turn, resumes on reply	docs

1. PROVIDED: B is handed the hidden fact, then writes the adapter; this is the capability ceiling.
2. COLD: B is alone, with no counterpart to answer; it may ask, but nothing comes back.
3. FORCED: B must commit from its own context and the sample and may not ask.
4. FORCED-FLAGGED: as FORCED, but the underspecified field is named while its convention is withheld, removing the detection burden.
5. DIALOGUE (VOLUNTEERS): A and B exchange up to three rounds and A may volunteer unrequested details, then B writes. B closes the exchange by replying READY.
6. DIALOGUE (ANSWERS-ONLY): as DIALOGUE (VOLUNTEERS), but A answers strictly what is asked; this isolates asking as the mechanism.
7. MITIGATION: as FORCED, but under a gated-commit rule; B must either establish every needed convention from its inputs or return an explicit abstention: the literal marker **ABSTAIN** followed by the missing question.

Which of these configurations do deployed pipelines implement? We audited eight code-agent harnesses in their documented default or unattended configuration, asking two questions of each. Does the agent have a tool whose effect is to put a question to a person and receive an answer? If it does, does the run block until the answer arrives? Table 3 summarizes the audit; the Supplementary Material gives the source file and line, or the documentation passage, behind each row. In their default unattended configuration, six of the eight harnesses give the agent no channel that blocks on a question (Table 3). Which of this study’s two no-channel configurations such a harness realizes depends on what its loop does with a turn that ends in a question rather than in code. If the run stops there, the harness is COLD, and the exposure is a stalled integration. If the loop prompts the agent to complete anyway, it is FORCED, and the exposure is a committed guess. The audit establishes the absence of the channel, not the loop’s reaction to a question; the two configurations bracket what such a harness ships.

3.4 Models and runs

We run 8 models: five local open-weight models spanning 2B–14B parameters (`gemma2:2b`, `qwen2.5:7b`, `llama3.1:8b`, `mistral-nemo:12b`, `qwen2.5:14b`) on consumer hardware; one API-served 70B open-weight model (`Llama-3.3-70B-Instruct-Turbo`) that fills the gap between the local tier and the frontier; and two frontier systems from different providers (Claude Opus 4.8 and GPT-5.6-sol) through their public APIs. Local and 70B models run 5 times per model–task pair (a *cell*); frontier models 3 times per cell, plus a dedicated COLD extension of seven further runs per irreducible task that is reported separately (Section 4.5) and excluded from every pooled rate. The realistic corpus (Section 3.2) was run on the five local models and on the two frontier systems, under four configurations (PROVIDED, COLD, FORCED, A-DOCS), with 5 runs per cell for the local models and 3 for the frontier systems. Decoding uses temperature 0.7 for the local and 70B models and provider-managed defaults for frontier models; the Supplementary Material gives the exact tags, hardware, and decoding, and every run’s recorded configuration ships with the artifact. All runs are recorded in a single frozen dataset.

3.5 Oracle and outcome definitions

Correctness is decided by code. Each task carries test cases split into *benign* cases, on which a wrong convention still passes, and *discriminating* cases, which depend on the missing fact. Each case yields one of three states: pass, wrong value, or raised exception. An outcome is a *silent failure* when the adapter runs on every case with no exception, passes every benign case, and returns a wrong value on at least one discriminating case. Any crash, refusal, or non-runnable output is loud and counted separately. Because any exception and any failed benign case is scored loud, the silent rate is a lower bound on the share of wrong adapters (Section 6). An explicit abstention under the MITIGATION rule is recorded as its own outcome. Generation is stochastic and then frozen; measurement is deterministic execution. This generate–measure cut makes every reported number re-derivable offline without a model in the measurement loop, and permits stricter transcript matchers to be applied to the frozen outputs.

3.6 Capability gating and statistics

To separate the phenomenon from coding weakness, a model enters the silent-failure claim only if its PROVIDED success rate clears 0.8; a stricter per-cell gate restricts to model–task cells where the model succeeded on every PROVIDED run of that task. Proportions carry Wilson 95% intervals. Because observations within a task share the same hidden fact, headline comparisons use task-cluster bootstrap confidence intervals: tasks, not runs, are the resampling unit. Analysis choices and directional predictions were pre-registered before the definitive runs, following the registered-reports practice [36]; the artifact includes the prediction file and its deviation log. Threat categories follow the standard experimentation framework [37].

3.7 Research questions

With the corpus, pipeline configurations, oracle, and gates in place, the study’s seven questions can be stated in terms of that machinery, ordered from measurement to cause to mechanism to intervention. Table 4 maps each question to the comparison that answers it and to its pre-registered prediction.

RQ1 (rate). **How often do capable LLM code agents produce silent failures when the reconciling fact is absent and asking is not possible?** The comparison against the PROVIDED ceiling on the same tasks separates what a model cannot build from what it silently builds wrong.

RQ2 (cause). **Is the failure driven by absent information, rather than by undetected ambiguity or by coding weakness?** FORCED-FLAGGED removes the detection burden while withholding the fact; the inferable controls and the per-cell capability gate remove coding weakness. What survives all three is commitment across absent information.

RQ3 (descriptive generality). **Does the phenomenon appear in both the single-convention SYN tasks and the interface- and industry-derived API tasks, and what do the failures look like?** The pre-registered exploratory SYN/API split (provenance of the API-derived tasks in Supplementary Table S1) descriptively checks whether the effect depends on task provenance; a qualitative reading of the generated adapters characterizes the recurring failure shapes.

RQ4 (scale). **Does the tendency to surface the missing fact track model scale?** If asking grew with capability, scale would buy groundedness indirectly; the cold ask-rates across the 2B-to-frontier range test that directly.

RQ5 (mechanism). **How much correctness does dialogue recover, and is asking or being volunteered the load-bearing channel?** A provider restricted to answering only what is asked isolates the integrator’s own question as the vehicle of the fact.

RQ6 (intervention). **Does a gated-commit rule reduce silent failure without collapsing completion where the convention is available?** The rule operationalizes the paper’s own prescription; the inferable controls quantify its cost.

RQ7 (realism). **Does the phenomenon appear when the mismatch, the documentation, and the tests come from real library version changes rather than from the authors, and does reading the counterpart’s documentation remove it?** The realistic corpus of Section 3.2 answers the first half under FORCED against PROVIDED; the A-DOCS configuration answers the second. Its predictions were written before the run and ship with the artifact.

4 Results

4.1 The finding at a glance

Table 5 puts the central finding in one place before the seven research questions take it apart. Each row is one capable model on the 17 irreducible tasks. The first column is what the model builds when handed the fact. The next two are what happens when

Table 4 Design map. Each research question with the comparison that answers it, the population it is estimated on, its outcome metric, and the expectation pre-registered before the definitive runs (P1–P7 in the artifact’s `predictions.md`, P8 in the realistic corpus’s prediction file; deviations logged there). *Capable* = models with PROVIDED success ≥ 0.8 ; *irred.* = irreducible tasks; *controls* = inferable tasks.

RQ	Focus	Comparison	Population	Metric	Pre-registered expectation
1	rate	FORCED vs PROVIDED	capable, irred.	silent failure	P1: silent ≥ 0.50 under FORCED
2	cause	FLAGGED vs FORCED; irred. vs controls; per-cell gate	capable	silent failure	P2: flagging shifts silent $\leq 10\text{pp}$; P3: irred. exceeds controls $\geq 20\text{pp}$
3	descriptive generality	SYN vs API under FORCED	capable, irred.	silent failure	P7: exploratory subset comparison
4	scale	COLD across model sizes	all models	ask-rate	P4: not monotone in size; frontier ≥ 0.8
5	mechanism	answers-only vs volunteers vs COLD	capable, irred.	success	P5: volunteers $\geq$ answers-only $>$ none; gap $\leq 10\text{pp}$ supports asking as the mechanism
6	intervention	MITIGATION vs FORCED	capable, irred. + controls	silent, abstention, success	P6: silent -20pp via abstention; control success within 15pp (over-abstention = named risk)
7	realism	FORCED vs PROVIDED, realistic corpus; irred. vs controls; A-DOCS vs FORCED	capable on that corpus	silent failure	P8: silent ≥ 0.50 on irred.; irred. exceeds controls $\geq 15\text{pp}$; A-DOCS $10-20\text{pp}$ below FORCED

the pipeline lets it ask but no one answers (COLD): whether it asks, and whether it commits a silently wrong adapter instead. The last is what happens when the pipeline requires a commitment and forbids the question (FORCED).

Table 5 The finding at a glance: the five capable models on the 17 irreducible tasks. Under COLD the agent may ask but no one answers; under FORCED it must commit without asking. Rows are ordered by PROVIDED success.

	Provided correct	Cold asks	Cold silent	Forced silent
Opus 4.8	1.00	0.98	0.00	0.59
GPT-5.6-sol	1.00	1.00	0.00	0.45
qwen2.5:14b	0.93	0.00	0.67	0.60
Llama-3.3-70B	0.81	0.00	0.62	0.64
qwen2.5:7b	0.76	0.01	0.34	0.74

Two things are visible at once. Reading down the COLD columns, the models split into two tiers: the two frontier systems ask on nearly every run and commit silently on none, while the three open-weight models almost never ask and commit silently instead. Reading across the frontier rows, the same two systems that never commit

Table 6 Headline comparisons for capable models.

Differences in percentage points with task-cluster bootstrap 95% intervals (tasks are the resampling unit; 10 000 draws); * marks intervals excluding zero. Estimates use the five capable models: irreducible-task rows draw on 357 runs per side (5 models, 17 tasks, per-model runs) and control rows on 126.

Comparison	Δ (pp)	95% CI
Forced vs Provided (silent, irreducible)	+55	[+40, +69]*
Flagged vs Forced (silent, irreducible)	+6	[+1, +12]*
Irreducible vs controls (Forced, silent)	+48	[+25, +68]*
API vs SYN (Forced, silent, irreducible)	+9	[-22, +38]
Volunteers vs answers-only (success, irred.)	-0	[-4, +3]
Answers-only vs Cold (success, irred.)	+61	[+52, +69]*
Mitigation vs Forced (silent, irreducible)	-55	[-72, -38]*
Mitigation vs Forced (success, controls)	-63	[-80, -47]*

silently when they may ask commit silently on about half of their runs when they may not, at unchanged capability. The failure moves with the wiring, not with the model. What the frontier systems’ asking does not buy is completion: with no one to answer, correctness across the five models is 9%, so an open channel that nobody staffs turns a wrong value into a stalled integration. The research questions that follow quantify each piece of this table in turn.

4.2 RQ1: rate of silent failure under forced commitment

Handed the fact, the five capable models build a correct adapter on 88% of runs; forbidden to ask, they return a silently wrong one on 62%. One case makes the mechanism concrete. In the date task, System A sends 03/05/2026 and means the 3rd of May, because its dates are day-first; Agent B, whose region writes month-first, reads it as the 5th of March and emits 2026-03-05, a well-formed ISO date and the wrong day. Nothing raises. On the shared sample 07/07/2026 the two conventions agree, so a test that exercises only the sample passes and the defect is invisible.

Figure 2 shows the outcome distribution of every configuration for the capable models on the irreducible gaps; Supplementary Table S3 reports every model, and Table 6 carries the cluster-interval estimate of every headline comparison in this section and the next.

Five of the eight models clear the capability bar (PROVIDED success ≥ 0.8): **qwen2.5:7b**, **qwen2.5:14b**, Llama-3.3-70B, Opus 4.8, and GPT-5.6-sol, spanning four model families.¹ Handed the reconciling fact, the five build a correct adapter on 88% of irreducible runs (Table 7): the task is within reach.

Withhold the fact and forbid the question, and the same models return a silently wrong adapter on 62% of runs, against 7% under PROVIDED (Tables 7 and 6). A stricter per-cell gate controls for task difficulty. Keeping only the model-task cells where the model built the adapter correctly on *every* PROVIDED run, so that its capability on that exact task is demonstrated, FORCED silent failure is again 62%. Where the model

¹The gate is on pooled PROVIDED success; **qwen2.5:7b** sits lower on the irreducible subset alone (Table 12). Dropping it leaves the effect intact: four models, PROVIDED 92%, FORCED silent 58%.

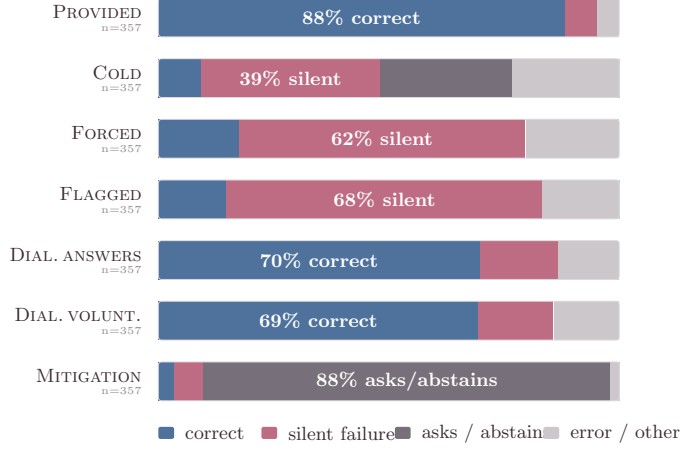


Fig. 2 Outcome by pipeline configuration for capable models (PROVIDED ≥ 0.8) on the irreducible tasks. Each bar is 100% of the runs of one configuration: **blue** correct, **rose** silent failure, **slate** asks or abstains, **grey** error/other.

Table 7 Outcome decomposition for capable models on the 17 irreducible tasks ($n = 357$ per configuration). Counts are mutually exclusive; loud comprises oracle errors and invalid output. These are the buckets drawn in Figure 2.

Configuration	Correct	Silent failure	Loud error	Asked/abstained	Other
Provided	315 (88.2%)	25 (7.0%)	17 (4.8%)	0 (0.0%)	0 (0.0%)
Cold	33 (9.2%)	139 (38.9%)	83 (23.2%)	102 (28.6%)	0 (0.0%)
Forced	63 (17.6%)	221 (61.9%)	72 (20.2%)	1 (0.3%)	0 (0.0%)
Flagged	53 (14.8%)	244 (68.3%)	60 (16.8%)	0 (0.0%)	0 (0.0%)
Dial. answers	249 (69.7%)	61 (17.1%)	47 (13.2%)	0 (0.0%)	0 (0.0%)
Dial. volunt.	248 (69.5%)	58 (16.2%)	50 (14.0%)	1 (0.3%)	0 (0.0%)
Mitigation	12 (3.4%)	23 (6.4%)	7 (2.0%)	315 (88.2%)	0 (0.0%)

demonstrably built the adapter with the fact, it fails silently without it: the failure tracks the absence of one fact rather than task difficulty.

A chance baseline sharpens the rate. On the twelve irreducible tasks that pose exactly two candidate conventions, a guesser choosing uniformly between them would be silently wrong on 50% of its commitments; the observed share is 78%. The models do not sample among the candidates. They concentrate on one convention, the quantitative counterpart of the confident default of Section 4.4.1. The candidate enumeration, the one finite-mapping task, and the four tasks with no enumerable candidate set are described in the Supplementary Material.

Answer to RQ1

Capable models handed the fact are correct 88% of the time; forbidden to ask and forced to commit, the same models produce a silent failure 62% of the time, and where their capability on the exact task is demonstrated, silent failure is again 62%.

Table 8 Silent failure on the SYN and API subsets: capable models, per subset. *Irred.* = irreducible tasks in the subset; control column reports silent failure on the subset’s inferable controls (count in parentheses).

Subset	Irred.	Provided (correct)	Forced (silent)	Flagged (silent)	Controls (silent)
SYN	5	0.89	0.55	0.64	0.05 (2)
API	12	0.88	0.65	0.70	0.19 (4)

4.3 RQ2: absent information, not detection or skill

Three controls separate absent information from its rivals: detection, skill, and recoverability.

First, detection. FORCED-FLAGGED names the underspecified field while withholding its convention, so the detection burden is removed. Silent failure does not fall. It rises to 68% (244 of 357 runs), six points above FORCED (task-cluster 95% CI +1 to +12). Naming the field also pushes the models toward writing code. They return a runnable adapter on 83% of runs, against 80% under FORCED. Among those adapters, the share that is silently wrong rises from 78% to 82%. Flagging the gap does not only make the committed adapters wrong more often; it also produces more of them.

Second, skill. The per-cell gate of Section 4.2 controls for per-task coding weakness: within the cells where the model built the adapter on every PROVIDED run, silent failure stays at 62%.

Third, recoverability. On the inferable controls, where the interface itself reveals the convention, the same models under the same FORCED order fail silently in only 14% of runs, against 62% on the irreducible tasks (Table 6). The failure concentrates where the fact is genuinely out of reach. Inferability is a matter of degree, though: two of the six controls, the alpha-3 country-code and decimal-comma tasks, still draw silent failure at 33% and 43%, because their convention leaves only a weak trace in the sample.

Answer to RQ2

Naming the field raises the silent rate by six points rather than lowering it. Inferable controls sit at 14% against 62% on irreducible gaps. Naming the gap neither supplies the absent fact nor prevents wrong commitment.

4.4 RQ3: descriptive generality across task subsets

Silent failure is 65% on the API-derived tasks and 55% on the SYN tasks, a difference of nine points whose task-cluster interval spans zero (Table 8); Figure 3 localizes the failure per task and model. This pre-registered exploratory comparison describes both subsets; it neither confirms a provenance effect nor establishes equivalence.

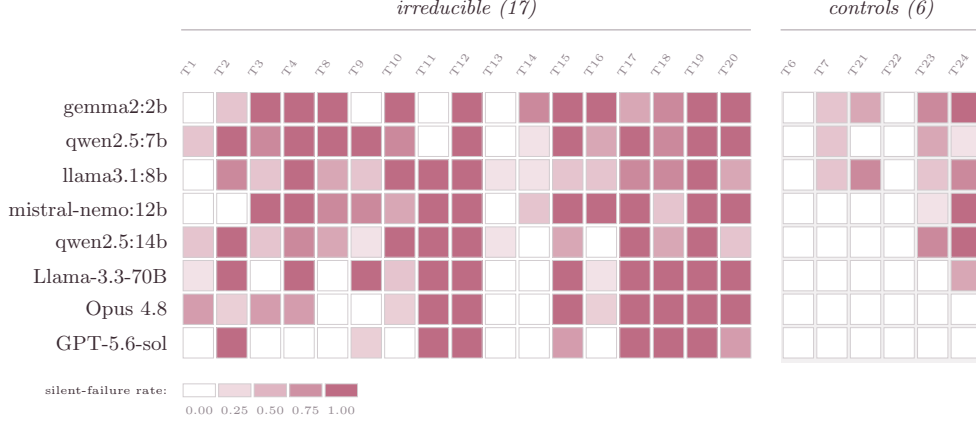


Fig. 3 Silent-failure rate under FORCED per model and task, from white (0) to full rose (1); an outlined white cell is a measured zero. Columns: the 17 irreducible tasks (left) and the 6 inferable controls (right, shaded); the one mixed task (T5) is omitted. Task codes are those of Table 1; the items the text discusses are T1 (date order), T14 (legacy country code) and T17 (spreadsheet date serial).

4.4.1 Qualitative analysis of failure modes

The frozen run retains the generated adapter and the full raw output for every run, which permits a qualitative reading of *what silent commitment looks like*. We read the 114 silent failures of the two frontier systems and give each one label, by a fixed precedence rule over lexical and structural cues. An adapter whose code states an assumption is an *acknowledged guess*. Otherwise, an adapter with mapping tables, input validation, or more than 24 lines is a *thorough wrong adapter*. Every other adapter is a *confident default*. The precedence makes the labels a partition even though the traces overlap.² Exemplar excerpts are hand-curated from the frozen record; the 144 explicit abstentions are read separately in Section 4.7, against each task’s *surfacing question* (the question about the hidden fact, fixed before the runs, that a well-aimed abstention should raise), where the gated-commit rule that produces them is analyzed.

Counts below cover the two frontier systems, where all three shapes occur; across the full run the 1,689 silent failures of all eight models are dominated in aggregate by the confident default (69%): it is the plurality shape for seven of the eight models. The exception is Opus, whose silent failures more often carry the assumption marker and commit regardless (Table 9).

Review traces do not transfer

An assumption-marker grep over a generated diff would flag 43 of Opus’s 62 silent failures (69%) and 0 of GPT’s 52 (Table 9). The two systems reach silent failure through different observable traces. Review tooling calibrated on the explicit comments of one therefore transfers poorly to the other.

²34 of the 114 frontier silent failures carry both an assumption marker and more than 24 code lines.

Table 9 Assumption-marker trace among all silent failures of the two frontier systems. The marker is the pre-specified lexical rule used by the qualitative classifier.

System	Marker	Silent	Share
Opus 4.8	43	62	69%
GPT-5.6-sol	0	52	0%

The confident default

46 of 114 silent failures (40%) are a clean, idiomatic adapter built on the integrator’s own convention, with nothing in the code signaling that a choice was made. GPT-5.6-sol concentrates here: 39 of its 52 silent failures (75%) take this shape. The entire committed adapter for the Fahrenheit telemetry task reads:

```
def adapt(a_payload):
    return {"celsius":
        round(float(a_payload["temp"]), 2)}
```

It passes every benign case and mislabels every real temperature.³

The thorough wrong adapter

25 silent failures (22%) show the opposite trace: defensive coercion, explicit mapping tables, input validation, edge-case comments. In one inventory-federation run the adapter defends against booleans, numeric strings, and malformed payloads across 36 lines, around a comment stating that System A’s `stock` convention is unavailable, and then encodes the wrong meaning for the absent field anyway. The thoroughness is real; the load-bearing assumption is still a guess, and the polish makes the defect harder to spot in review.

The acknowledged guess

43 silent failures (38%) carry an explicit assumption marker and commit regardless, and the shape is model-specific: almost all are Opus 4.8’s and none GPT-5.6-sol’s (Table 9). A spreadsheet-serial adapter opens with the comment “Interpret the integer day count as days since the Unix epoch (1970-01-01), per our own tooling’s convention” and proceeds to a clean, wrong conversion. The gap is detected, written down, and crossed. The two frontier systems thus fail silently with opposite traces at nearly the same rate, one documenting the guess and one leaving no trace; this is direct qualitative evidence for the RQ2 result that, once asking and abstention are removed, detecting the gap does not avert the commitment.

³The confident/thorough split is not an artifact of the 24-line cutoff: at cutoffs of 18 and 30 lines the confident share stays at 39–40%.

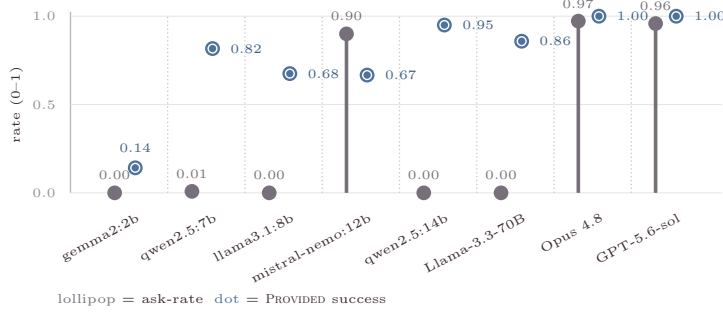


Fig. 4 Asking does not track scale. Per model, ordered by size: **slate** lollipops give the cold ask-rate (how often the model asks instead of committing when alone); **blue** dots give PROVIDED success, the capability ceiling.

Answer to RQ3

The exploratory subset rates are 65% on API-derived tasks and 55% on SYN tasks (+9pp, cluster CI -22 to $+38$), with no detectable difference and no equivalence claim. Qualitatively, silent failures are dominated by the confident default and, in the frontier systems, by the thorough and the acknowledged variants: polish, documentation, and error co-occur.

4.5 RQ4: asking does not track scale

The cold ask-rate ranges from 0% to 90% across the six open-weight models and does not rise with capability (Figure 4). If asking grew out of capability, the ask-rate would rise with PROVIDED success; in the eight sampled models it does not, the pattern our pre-registration predicted (P4). The dissociation is a measured fact; what eight models from different families cannot settle is its *cause*, size versus instruction tuning, alignment, or provider defaults, so we hold the causal reading as exploratory. The best local builder asks on 0% of cold runs and a weaker one on 90%; the two frontier systems combine both competences, asking on 96–97% of cold runs over all 24 tasks. Under the FORCED order their requests are suppressed but not eliminated, which is why their silent rates are floors. The two competences came apart in our sample, which is why a pipeline cannot assume one from the other.

The COLD configuration is itself a deployment scenario: an agent that may ask but is wired to a partner that never responds. Figure 2 puts its silent-failure bar at 39% for the capable models. That aggregate averages two opposite populations (Table 5). The two frontier systems ask on nearly every irreducible run and commit a silent failure on *none*; counting the dedicated COLD extension of Section 3.4, the count is 0 of 170 runs for each, with a 95% upper bound of 2.2%. The three open-weight capable models ask on 0–1% of runs and commit silently on a third to two thirds of them. An open channel is therefore not sufficient: a model that does not spontaneously surface the gap commits silently even when it could have asked, and for the majority of our capable models the cold failure is a silent commit, not an unrouted question. The channel has to be one the pipeline requires the agent to use.

Answer to RQ4

Cold ask-rates span 0% to 90% across local scales with no monotonic trend. On the irreducible tasks the two frontier systems ask on nearly every cold run and commit silently on none; the three capable open-weight models almost never ask and commit silently instead. An open channel is not enough, because a model that does not spontaneously ask does not use it.

4.6 RQ5: dialogue recovery and its mechanism

Alone, with no counterpart, capable models solve 9% of irreducible runs: this is the COLD configuration of Figure 2. Given up to three rounds with the provider, correctness rises to 70% when the provider answers only what is asked and to 69% when it may volunteer unrequested details (Table 7). Dialogue remains a partial defense: 70% stands well below the 88% PROVIDED ceiling, and 17% of dialogue runs still end in silent failure.

The mechanism is sharper once the dialogue actually happens. In about a quarter of the runs the integrator declared **READY** on turn one, so the provider never spoke and the volunteers configuration could not act; those runs mostly fail.⁴ Restricting to the runs where the integrator did ask, correctness is 86% for answers-only and 87% for volunteers, within one point and far above the COLD baseline. The pre-registration committed to reading a gap of at most 10 points between the two dialogue configurations as support for asking as the recovery mechanism; the observed gap is one point (Table 6). It does not test volunteering in the runs where no provider utterance occurred.

Answer to RQ5

Dialogue recovers correctness to 70% against an 88% ceiling, and answering only what is asked recovers as much as volunteering; the mechanism is the integrator’s own question. Volunteering remains undertested when the provider never speaks.

4.7 RQ6: the gated-commit rule

The gated-commit rule cuts silent failure from 62% to 6%, and cuts completion on the inferable controls from 81% to 18%. Figure 5 shows the trade-off directly.

On the irreducible tasks, silent failure falls to 6%, a drop of 55 points (Table 6). The rule converts silent commits into explicit abstentions on 88% of runs.⁵

The completion check fails, exactly as the pre-registered risk anticipated. On the inferable controls, where the interface holds the answer, **FORCED** models succeed in 81% of runs. The gated models abstain in 72% and succeed in only 18%, a completion collapse of 63 points.

The abstentions are on point. Matched against each task’s pre-registered surfacing question (the same keyword proxy used for dialogue surfacing), 80% of irreducible-task abstentions across all models name the hidden mismatch, and among the frontier

⁴85 of 357 runs per dialogue configuration; 14 of them correct under answers-only and 12 under volunteers.

⁵Two further runs ask a question without the formal **ABSTAIN** marker (both **qwen2.5:7b**), which yields the asked-or-abstained count of Table 7.

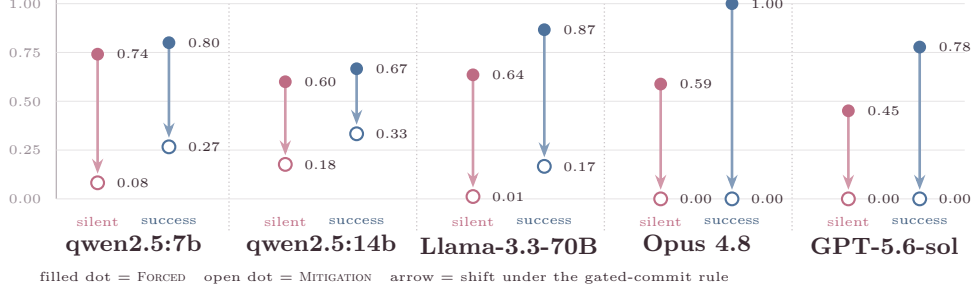


Fig. 5 The gated-commit trade-off per capable model, on a shared 0–1 axis. Filled dot: FORCED; open dot: MITIGATION; the arrow is the shift the rule induces. **Rose**, silent failure on irreducible gaps: the drop is the safety gain. **Blue**, success on inferable controls: the drop is the completion cost.

systems this rises to 93%: the payments abstention asks whether **amount** is in minor units and which per-currency exponent applies. These are appropriate questions with no answering channel. The over-abstention on controls has the same character: on the bounded alpha-3 task the abstention reads “is ‘ESP’ actually alpha-3, or could it be a non-standard/custom code? I cannot confirm the encoding convention from a single sample”, declining a question the interface had already answered (verbatim model output from the frozen record); the frontier systems’ control abstentions almost always name the right field while declining to act on it. The over-abstention is therefore a property of the rule’s threshold rather than of detection: the models identify the field whose convention the interface settles, and still treat the inference as a guess because the rule asks for a convention to be *established*. A deployable gate needs a second clause that admits conventions the sample determines.

The rule sets a high bar, admitting code only when every convention is *established* rather than merely plausible; under it, a model abstains even where the interface makes the convention recoverable, so the completion cost is real whether or not each single abstention is judged an error. The rule as stated does not separate an absent fact from an inferable one; it buys safety with completion.

Answer to RQ6

The rule cuts silent failure from 62% to 6% on irreducible gaps by converting commits into explicit abstentions (88%); on inferable controls completion collapses by −63pp (81% to 18%). Safety is bought with completion: the pre-registered over-abstention risk materialized.

4.8 RQ7: the realistic corpus

On tasks whose mismatch, documentation, and tests come from real library version changes, the three models that clear the capability bar fail silently on between 27% and 60% of forced runs over the irreducible tasks. Table 10 carries every rate of this section: the five local models and the two frontier systems under the four configurations run on this corpus, with the number of runs behind each cell. Three systems clear the

PROVIDED bar here, **qwen2.5:14b**, GPT-5.6-sol, and Opus 4.8. They are a corpus-specific set, not the five capable models of the authored corpus, and the rates that follow are theirs. Five irreducible tasks, with 25 runs per configuration for the open-weight model and 15 for each frontier system, are a small base, and the intervals in Table 11 say so.

Table 10 The realistic corpus (5 irreducible tasks, 7 controls). Rates on the irreducible tasks unless marked: capability ceiling, whether the model asks under COLD, silent failure under COLD, FORCED and A-DOCS, loud failure under FORCED (the adapter ran but broke a benign case, or was invalid), silent failure on the controls under FORCED, and n , the runs per configuration on the irreducible tasks. A-DOCS is FORCED with the old version’s published documentation added to Agent B’s context. † : below the PROVIDED ≥ 0.8 bar on this corpus.

	Provided	Cold		Forced		A-docs	Controls	
	correct	asks	silent	silent	loud	silent	silent	n
Opus 4.8	1.00	1.00	0.00	0.27	0.20	0.40	0.19	15
GPT-5.6-sol	1.00	0.53	0.13	0.40	0.20	0.20	0.38	15
qwen2.5:14b	0.96	0.00	0.32	0.60	0.08	0.44	0.11	25
mistral-nemo:12b †	0.76	0.92	0.00	0.32	0.52	0.32	0.20	25
qwen2.5:7b†	0.60	0.00	0.48	0.24	0.60	0.24	0.17	25
llama3.1:8b †	0.48	0.00	0.28	0.36	0.56	0.16	0.37	25
gemma2:2b†	0.04	0.04	0.48	0.56	0.44	0.56	0.31	25

The three capable systems tell three different stories. The open-weight model replicates the authored corpus: high silent failure on the irreducible tasks, low on the controls, no questions asked. GPT-5.6-sol replicates the forced rate but not the rest: it is nearly as wrong on the controls as on the irreducible tasks, and it asks on only half of its cold runs. Opus 4.8 keeps its authored behavior on the open channel, asking on every cold run and committing silently on none, but the counterpart’s documentation makes it worse rather than better.

qwen2.5:14b, forbidden to ask, returns a silently wrong adapter on 60% of irreducible runs and on 11% of the controls: high on the irreducible tasks, low on the controls, the same shape the authored corpus showed, now on material the authors did not write. Per task the spread is wide: silent failure is total on the language-codes task (R02), high on the scale-word (R01) and unit-base (R06) tasks, and zero on the symbol-ascii task (R03), where the model’s own default happens to coincide with the new convention. With the old version’s published documentation in hand (A-DOCS) its silent failure falls to 44%, inside the band predicted before the run; where the change is visible in the documentation, as for the unit base, it drops to zero, and where the documentation of the two versions reads the same, as for the scale word, it does not move. Left free to ask, it never asks; its COLD silent rate is lower than under FORCED, but the difference turns into loud failures that break a benign case (36% of cold runs), not into correct adapters, so the prediction that COLD would sit within 10 points of FORCED fails in the direction of more visible error.

GPT-5.6-sol fails silently on 40% of forced irreducible runs, against 45% on the authored corpus, but also on 38% of the controls, so the gap between irreducible tasks and controls that the open-weight model shows does not appear for it. Left free to ask, it asks on 53% of runs, not the 96–98% of the authored corpus. The counterpart’s documentation lowers its silent failure to 20%; the failures that remain are all on the scale-word task, whose documentation reads the same in both versions. Two of the four predictions written before this run therefore failed, both in the same direction: on real tasks this frontier system asks less, and is wrong on the controls more, than the authored corpus predicted.

Opus 4.8 asks on every cold run and commits silently on none, as on the authored corpus; nothing is delivered either, since no one answers. Forbidden to ask, it fails silently on 27% of irreducible runs, below its 59% on the authored corpus and below the band predicted before the run, and the gap to the controls is eight points. With the counterpart’s documentation in hand it fails silently *more* often, on 40% of runs: every failure is on the scale-word and language-codes tasks, where the old version’s documentation describes the convention the adapter is supposed to translate away from, and the model keeps it. The prediction that the documentation would help therefore failed in the opposite direction. Across the three capable systems, the counterpart’s documentation helped one, barely helped another, and hurt the third; the change usually lives in the changelog.

Answer to RQ7

On 12 tasks built from real library version changes, the three capable systems fail silently on between 27% and 60% of forced runs over the irreducible tasks. Left free to ask, Opus 4.8 asks on every run, GPT-5.6-sol on half, and the open-weight model never. Reading the counterpart’s documentation lowers silent failure for two systems and raises it for the third: the convention change lives in the changelog, and documentation of the old version can anchor a model to the old convention.

5 Discussion

Table 11 closes the pre-registration loop before interpretation: each prediction, locked before the definitive runs, against what the runs returned.

5.1 What the numbers describe

Across repeated runs in the frozen record, a model first produced a correct adapter when its input included one resolving sentence. When the next configuration omitted that sentence but still required a commitment, the model produced clean, confident, well-commented code. The code ran. It passed the tests that resembled the sample. And it was wrong, in the one place that mattered, for the one reason nothing in the code could reveal: the fact that reconciles the two systems was never in the room. That is the whole phenomenon, and it is worth stating plainly because its two halves usually get separated. The competence is real: 88% correct with the fact in hand. The failure is real: 62% silently wrong without it. Both are the same model. Capability and groundedness are different axes, and the clearest evidence is within a single model:

Table 11 Pre-registered predictions against outcomes. Predictions were locked before the definitive runs; the deviation log in the artifact records every post-lock mechanics fix.

	Prediction (abridged)	Outcome	Verdict
P1	Capable models commit silently on $\geq 50\%$ of forced irreducible runs	0.62 [0.46–0.76]	confirmed at point estimate ^d
P2	Flagging the gap shifts the silent rate by $\leq 10\text{pp}$	+6pp [+1,+12]	magnitude held; sign not predicted ^a
P3	Irreducible tasks exceed inferable controls by $\geq 20\text{pp}$	+48pp [+25,+68]	confirmed
P4	Ask-rate not monotone in model scale; frontier ask-rate ≥ 0.8	0–90% locals; 0.97/0.96	confirmed
P5	Recovery: volunteers $\geq$ answers-only > no dialogue	gap –0pp [–4,+3]; +61pp over COLD	consistent; volunteering unexercised ^b
P6	Gated commit: silent –20pp, completion cost within 15pp	silent –55pp; controls –63pp	safety met; completion cost hit ^c
P7	SYN vs API difference (exploratory, no direction)	+9pp [–22,+38]	no detectable difference
P8	Realistic corpus: silent ≥ 0.50 on irred. for the capable model; irred. – controls $\geq 15\text{pp}$; A-DOCS 10–20pp below FORCED	open-weight: 0.60 [0.41–0.77]; +49pp; –16pp. GPT-5.6-sol: 0.40 [0.20–0.64]; +2pp; –20pp. Opus 4.8: 0.27 [0.11–0.52]; +8pp; +13pp	confirmed (open-weight); controls not separated (frontier); A-DOCS hurt Opus ^e

^athe $\leq 10\text{pp}$ magnitude held, but the sign was not predicted: the 95% interval lies entirely above zero, so flagging *raises* the rate rather than preserving it. ^bthe $\leq 10\text{pp}$ gap between the two dialogue configurations is consistent with asking as the mechanism; in 85 runs per configuration the provider never spoke, so volunteering was not exercised. ^csilent reduction met; the completion bound failed exactly as the pre-registered over-abstention risk anticipated. ^dthe point estimate (0.62) clears the pre-registered 0.50 threshold, but the task-cluster interval’s lower bound (0.46) falls below it: confirmed at the point estimate, not established at the 95% level. ^eseven auxiliary predictions, each written before its own run, did not hold: two for the open-weight model, two for GPT-5.6-sol and three for Opus 4.8. The expected and observed values are in the Supplementary Material.

the two frontier systems build the adapter perfectly when given the fact, yet fail silently without it on 59% (Opus 4.8) and 45% (GPT-5.6-sol) of forced runs over the irreducible tasks, and what recovers the failure is asking rather than more capability. Buying more of the first does not supply the second. The named models appear here as capable instances of the phenomenon, not as objects of comparison; the finding concerns the capable models we studied, not which system ranks higher.

5.2 Where reliability has to live

If capability cannot close the gap, something else has to, and our configurations sketch the space of candidates. Dialogue works when it happens: the answers-only configuration shows that the naked act of asking, with no volunteered help, lifts correctness by 61 points against an 88% ceiling (Section 4.6). A commitment gate works differently: it converts silent wrongness into loud abstention, cutting silent failure by 55pp, and it overshoots, abstaining on 72% of the inferable controls where the interface held the answer. There is a lesson in the overshoot. The naive rule “never commit across a gap” is not free; it trades completion for safety at a rate that depends on how reliably the pipeline distinguishes an absent fact from an inferable one. Reliability engineering for agent integration will live in that discrimination: in process rules that route

absent knowledge toward a counterpart, a validator, or a person, while letting inferable conventions through. The safeguard belongs to the pipeline, and a pipeline should not assume that selecting a more capable model will also supply reliable clarification. Expecting that from scale is, on this evidence, misdirected.

5.3 Delegation and the classic problem

Interoperability is one of software engineering’s oldest chores, and it has traditionally been caught by people noticing things: a reviewer who squints at a date format, an integration test that fails loudly on the staging server, an engineer who emails the other company. Our results motivate a test of whether delegation removes those human checks; this study is not that comparison. A human handed only B’s specification and one sample would also have to guess and might project a convention.

The classic problem does not disappear; its failure mode changes shape. A loud incompatibility at the interface becomes a quiet wrong value that passes the common case, and the cost moves downstream, into ledgers and dashboards and decisions, where it no longer looks like an integration error at all; the 62% silent rate under FORCED is that shift made quantitative. What delegation cannot remove is the source of the missing fact. When the fact cannot be standardized into both agents’ inputs beforehand, someone or something still has to supply it, as the dialogue configurations’ partial recovery shows. The loop can be redrawn. It cannot be emptied.

5.4 Relation to the specification gap

This study composes with the specification-gap result on coordination under partial knowledge [34] into a single picture with two situations. When two agents share one incomplete specification of one artifact, restoring the specification restores coordination: the missing knowledge exists, centrally, and can be written back in. When the specification is split across organizations, there is nothing to restore. The reconciling fact lives on the far side of a boundary, and the only moves are to cross the boundary or to guess. The first situation rewards specification-first engineering; the second rewards commitment discipline: rules about what an agent may do when its inputs cannot settle a question. The numbers rhyme across the two studies: restoring a shared specification recovers 89% there, and being handed the missing fact recovers 88% here; what differs is that here no local repair can substitute for the fact crossing the boundary. Together they suggest that coordination failure in code agents is not one problem but two, split by where the missing knowledge sits. What this study adds to that picture is empirical: the pipeline’s handling of the agent’s question is the manipulated variable, so the same models are measured with the channel open, closed, and gated; the harness audit ties the closed configuration to shipped tools; and the realistic corpus shows the effect on mismatches the authors did not write.

5.5 Implications for practice

The results translate into concrete guidance for teams that deploy code agents on integration work.

Table 12 Per-model silent-failure rate under FORCED, on the irreducible tasks and on the full corpus, with each model’s PROVIDED success on the irreducible tasks. Rows are ordered by capability; [†] marks models below the pooled PROVIDED ≥ 0.8 bar.

Model	Provided (irred.) correct	Forced silent (irred.) silent	Forced silent (full corpus) silent
Opus 4.8	1.00	0.59	0.42
GPT-5.6-sol	1.00	0.45	0.32
qwen2.5:14b	0.93	0.60	0.50
Llama-3.3-70B	0.81	0.64	0.47
qwen2.5:7b	0.76	0.74	0.57
llama3.1:8b [†]	0.69	0.62	0.54
mistral-nemo:12b [†]	0.64	0.71	0.55
gemma2:2b [†]	0.08	0.68	0.60

First, *capability reduces but does not remove the failure*. Silent failure trends downward across the five capable models (Table 12), a descriptive ordering over five models with no uncertainty on the model axis, not a fitted law. At the top the trend stops discriminating: the two systems that reach a perfect 100% PROVIDED ceiling still fail silently on 59% (Opus 4.8) and 45% (GPT-5.6-sol), a 14-point gap at saturated capability that capability cannot explain. Screening models by their score on this corpus shrinks the failure but does not remove it, because the fact the adapter needs is absent from the integrator’s specification, not from the space of things a model can do.

Second, *the defense belongs in the process, not the prompt*. Naming the underspecified field for the agent while still requiring it to commit is counterproductive in our data: it raises the silent rate by 6 points. Giving the agent a channel to ask, and a counterpart that answers, recovers most of the ceiling. Integration pipelines should route an agent’s questions to a human or to the partner system, not assume the agent will refuse to guess.

Review automation should cover multiple failure traces. An assumption-comment rule catches most of Opus’s silent failures and none of GPT’s (Table 9), so a grep calibrated on one system is not a portable safety check. Execution on discriminating cases remains the common check.

Third, *a naive abstention rule is not free*. Gating commitment behind explicit abstention nearly removes silent failure (to 6%) but collapses completion on the tasks whose convention was actually available, from 81% to 18%. A deployable gate must discriminate an absent fact from an inferable one, for example by attempting retrieval or a validation query before abstaining. As stated the rule is not deployable: a 63-point drop in completion would stall a production pipeline. What the result motivates is an architecture around the rule: a staged human-in-the-loop that escalates only the abstentions, a cross-checking validator agent that resolves the inferable cases automatically, or a retrieval step that supplies documented conventions before the gate fires. The gate is a signal to route, not a place to stop.

6 Threats to validity

6.1 Construct validity

The silent-failure construct is deliberately conservative: an adapter that raises *any* exception, or that fails *any* benign case, is counted as loud rather than silent, so the reported silent rates are lower bounds. A silent failure is thus an adapter that executes without exception and returns a wrong value on a discriminating case; for the nine tasks whose cases are all discriminating there is no benign case to pass, so “silent” there means executes-clean-and-wrong without the common-case demonstration. Excluding those nine leaves the headline the same (62% either way), so the effect is not an artifact of their construction.

The keyword-based surfacing measure on transcripts is a stated proxy; it can miss paraphrases and fire on incidental mentions, and it is measured independently of adapter success. Abstention detection is lexical (an **ABSTAIN** marker in the reply’s head) and the on-point reading of an abstention reuses the same keyword proxy; both are recorded in the frozen run, so stricter matchers can be re-applied without re-generation.

To quantify the proxy, two large language models of different lineage (a Claude model and a GPT model) labelled a stratified sample of 60 abstentions for whether the text raises the task’s hidden mismatch, under a criterion set and reviewed by the author. This is a second reading of a *content* question, kept separate from the oracle, which alone decides correctness. Agreement between the two readers was moderate (Cohen $\kappa = 0.59$, Gwet AC1 = 0.67). Against their consensus labels the keyword matcher reached precision 0.89 and recall 0.89; the naive recall over-weights the stratum where the misses concentrate, and reweighting to the population ratio raises recall to 0.93. The proxy is accurate but imperfect, and its residual misses are paraphrases that engage the convention without the exact term, so the on-point rates of Section 4.7 carry roughly this classification noise. The sampling design and the confidence intervals are in the Supplementary Material. Annotation at scale, and by human raters, is left to future work.

6.2 Internal validity

Whether a model asks may be prompt-sensitive rather than a stable property. The FORCED instruction does not fully suppress asking in frontier models; their silent rates are therefore floors. The 62% FORCED rate is conditional on an instruction that requires completion without asking; it estimates the failure of pipelines that force a commit despite a missing fact, not an unconditional agent error rate. Runs are stochastic samples at temperature 0.7 (local and 70B) and provider-managed defaults (frontier); we read direction and magnitude, not two-digit precision. All configuration results rest on one prompt template per configuration. A paraphrase check on the two local capable models reworded the FORCED and MITIGATION prompts twice, keeping the instructions, and re-ran the 17 irreducible tasks (Supplementary Table S4). The pre-registered band was 10 points. FORCED silent failure stayed inside it under both rewordings, moving by at most nine points and in both directions. The gate’s

abstention rate did not: one rewording raised `qwen2.5:14b`’s abstention from 0.72 to 0.86, with silent failure falling accordingly. The direction of the MITIGATION trade-off is stable across wordings and its size is not, so the abstention and silent rates of Section 4.7 are one wording’s values. The COLD prompt was not reworded and the frontier models were not re-run.

6.3 External validity

The corpus is built to isolate one missing fact per task, which is what makes the oracle deterministic; it is not a sample drawn from field integrations, and every rate in this paper is a rate on this corpus. It covers one family of mismatch: scalar and format conventions (units, scales, encodings, code maps). Integration failures of other shapes lie outside it, among them stateful protocols and authentication flows (for example OAuth token exchange), asymmetries between deeply nested JSON schemas, and semantic drift in enumerated vocabularies; whether the same silent-commit behavior holds there is untested.

The API subset reproduces documented conventions of real interfaces, and the oracle executes each generated adapter against those documented conventions rather than against the live service. That choice keeps every measurement deterministic and re-derivable offline; what it leaves unobserved is service-side behavior such as undocumented drift. The realistic corpus removes the authors from the mismatch but not from the selection, whose funnel Section 3.2 reports stage by stage. The capability admission favors conventions expressible as a short rule over those that live in a lookup table, but it removed one task; the filter that shaped the corpus is the existence of a silent divergence, which is a property of the libraries. Its rates are rates on those 12 tasks, five of them irreducible, with 15 to 25 runs per configuration and model.

The realistic corpus is drawn from public libraries, so the new conventions may appear in the models’ training data. The new versions behind the ten task-bearing pairs were released between December 2024 and August 2026, seven of them in 2026 (the artifact records both release dates per pair); the training cutoffs of the frontier models are not published, so we cannot separate the pairs that postdate them. A memorized convention would show as a lower FORCED silent rate on the realistic corpus than on the authored one. Opus 4.8 is lower there and GPT-5.6-sol is not (Tables 10 and 12), and the open-weight model, whose training predates every 2026 release, matches its authored rate; the pattern is compatible with either partial memorization or easier tasks, and the corpus is too small to tell them apart.

Rates are conditional on the pipeline configuration and cluster by task; the task-cluster intervals reflect that. The bootstrap resamples the tasks in this purposively built corpus, so the intervals quantify sensitivity to that task set rather than generalization to a task population; the eight models are treated as fixed, and statements about “capable agents” are about the five sampled capable models, not a random draw. A provider that emits its brief before any integrator turn would test volunteering directly, including the 85 runs per dialogue configuration in which the provider did not speak.

6.4 Conclusion validity

All headline comparisons, and the headline rates, carry task-cluster bootstrap intervals that resample tasks and so account for within-task correlation; the per-rate intervals elsewhere in the text are Wilson binomial and, treating runs as independent, are narrower than a task-cluster interval, so they index within-sample precision rather than generalization. Per-task heterogeneity is visible directly in the model-task map of Figure 3 and, as rates, in the per-task breakdown of Supplementary Table S2. With only 17 task clusters, the cluster bootstrap is known to be anticonservative [38]; its intervals should be read as indicative of task-set sensitivity, not exact coverage. Several irreducible tasks (the legacy-country and date-order items among them) sit well below the headline rate, which Supplementary Table S2 makes inspectable; the effect is a distribution over tasks, not a constant. Predictions were locked before the definitive runs, and deviations are logged in the artifact.

T17 is a boundary case: PROVIDED success is 0.48 while FORCED silent failure is 1.00. Capable models can fail this task even with the fact, so the per-cell gate excludes its undemonstrated cells from the demonstrated-capability claim.

7 Conclusion

Add the missing fact to a capable code agent’s input and it builds the integration: 88% correct across five capable models from four families. Omit the fact, and what happens next depends on the pipeline, not on the model. Where the pipeline lets the agent ask, the two frontier systems ask on nearly every run and never commit silently, but the integration stalls unless someone answers. Where the pipeline requires a commitment and forbids the question, the five capable models return an adapter that runs clean and returns the wrong value on 62% of runs, and the pattern survives when the mismatch, the documentation, and the tests come from real library version changes. Asking is worth 61 percentage points when a counterpart answers, and the tendency to ask does not rise with scale. The safeguard is therefore a property of the wiring: a channel that routes the agent’s question to the partner system or to a person, and tests on the cases that discriminate between conventions. A commit-gating rule that turns absent knowledge into explicit abstention is a measurable step toward that wiring, at a completion cost the pipeline must then manage.

Acknowledgements. During the preparation of this work the author used LLM-based assistants to help implement the experiment scripts and to edit the manuscript, under the author’s direction and verification. The author reviewed all content and takes full responsibility for the paper.

Data availability. The corpus, deterministic oracle, harness, frozen runs, pre-registered predictions, and the analysis script that regenerates every reported number (offline, no models, seeded so confidence intervals reproduce exactly) are archived on Zenodo at <https://doi.org/10.5281/zenodo.22735792>. The Supplementary Material (Online Resource 1: Figure S1, Tables S1–S4, the pipeline-configuration prompts, and the provenance of every realistic-corpus task) accompanies this article as a separate PDF.

References

- [1] Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys* **22**(3), 183–236 (1990) <https://doi.org/10.1145/96602.96604>
- [2] Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. *The VLDB Journal* **10**(4), 334–350 (2001) <https://doi.org/10.1007/s007780100057>
- [3] Parnas, D.L.: On the criteria to be used in decomposing systems into modules. *Communications of the ACM* **15**(12), 1053–1058 (1972) <https://doi.org/10.1145/361598.361623>
- [4] Meyer, B.: Applying “design by contract”. *Computer* **25**(10), 40–51 (1992) <https://doi.org/10.1109/2.161279>
- [5] Mens, T.: A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering* **28**(5), 449–462 (2002) <https://doi.org/10.1109/TSE.2002.1000449>
- [6] Dig, D., Johnson, R.: How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution: Research and Practice* **18**(2), 83–107 (2006) <https://doi.org/10.1002/smr.328>
- [7] Ehtesham, A., Singh, A., Gupta, G.K., Kumar, S.: A survey of agent interoperability protocols: Model context protocol (MCP), agent communication protocol (ACP), agent-to-agent protocol (A2A), and agent network protocol (ANP). *arXiv preprint arXiv:2505.02279* (2025) [arXiv:2505.02279](https://arxiv.org/abs/2505.02279)
- [8] Mu, F., Shi, L., Wang, S., Yu, Z., Zhang, B., Wang, C., Liu, S., Wang, Q.: ClarifyGPT: A framework for enhancing LLM-based code generation via requirements clarification. *Proceedings of the ACM on Software Engineering* **1**(FSE), 2332–2354 (2024) <https://doi.org/10.1145/3660810>
- [9] Wu, J.J., Fard, F.H.: HumanEvalComm: Benchmarking the communication competence of code generation for LLMs and LLM agent. *ACM Transactions on Software Engineering and Methodology* (2025) <https://doi.org/10.1145/3715109>
- [10] Fang, Z., Jin, D., Dong, Y., Li, Y., Zhang, K., Jin, Z., Li, G.: ClarifyCodeBench: Evaluating LLMs on clarifying ambiguous requirements for code generation. *arXiv preprint arXiv:2607.00711* (2026) [arXiv:2607.00711](https://arxiv.org/abs/2607.00711)
- [11] Vijayvargiya, S., Zhou, X., Yerukola, A., Sap, M., Neubig, G.: Ambig-SWE: Interactive agents to overcome underspecificity in software engineering. *arXiv preprint arXiv:2502.13069* (to appear, ICLR 2026) (2025) [arXiv:2502.13069](https://arxiv.org/abs/2502.13069)

- [12] Li, J., Wu, Y., Chang, Y.: ClarEval: A benchmark for evaluating clarification skills of code agents under ambiguous instructions. arXiv preprint arXiv:2603.00187 (2026) arXiv:2603.00187
- [13] Peng, R., Ma, Z., Pang, A., Li, S., Zhang Xi-Jia, Yu, Y., Bara, C.-P., Chai, J.: Communication and verification in LLM agents towards collaboration under information asymmetry. arXiv preprint arXiv:2510.25595 (2025) arXiv:2510.25595
- [14] Wang, C., Yang, Y., Du, B., Lin, S., Liu, H.: LLM agents for deliberative collaboration: A study on joint decision making under partial observability. arXiv preprint arXiv:2607.06157 (2026) arXiv:2607.06157
- [15] Yao, Y., Zou, C., Hawkins, R.D.: Talk is cheap, communication is hard: Dynamic grounding failures and repair in multi-agent negotiation. arXiv preprint arXiv:2605.01750 (2026) arXiv:2605.01750
- [16] Li, Y., Naito, A., Shirado, H.: Systematic failures in collective reasoning under distributed information in multi-agent LLMs. arXiv preprint arXiv:2505.11556 (to appear, ICML 2026) (2025) arXiv:2505.11556
- [17] Chen, M., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021) arXiv:2107.03374
- [18] Austin, J., et al.: Program synthesis with large language models. arXiv preprint arXiv:2108.07732 (2021) arXiv:2108.07732
- [19] Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.R.: SWE-bench: Can language models resolve real-world GitHub issues? In: International Conference on Learning Representations (ICLR) (2024)
- [20] Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L., Lou, Y.: Large language model-based agents for software engineering: A survey. *ACM Transactions on Software Engineering and Methodology*, 3796507 (2026) <https://doi.org/10.1145/3796507>
- [21] Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S.K.S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., Schmidhuber, J.: MetaGPT: Meta programming for a multi-agent collaborative framework. In: International Conference on Learning Representations (ICLR) (2024)
- [22] Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., Sun, M.: ChatDev: Communicative agents for software development. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186 (2024). <https://doi.org/10.18653/v1/2024.acl-long.810>
- [23] Aytekin, M.C., Yilmaz, F.G., Demirezen, M.U.: Automating code generation for

- a new ecosystem: establishing baselines with large language model based code generation for ArkTS and HarmonyOS. *Automated Software Engineering* **33**(2), 59 (2026) <https://doi.org/10.1007/s10515-026-00599-9>
- [24] Ananbeh, O., Alnozami, W., Kim, D.-K.: Assessing the effectiveness of large language models for Java vulnerability repair: A comparative study. *Automated Software Engineering* **33**(3), 75 (2026) <https://doi.org/10.1007/s10515-026-00622-z>
 - [25] Cai, J., Li, J., Li, N., Mao, Z., Zhang, M., Tei, K.: SAGE: Semantic-aware gray-box game regression testing with large language models. *Automated Software Engineering* **33**(3), 102 (2026) <https://doi.org/10.1007/s10515-026-00635-8>
 - [26] Xu, J., Lu, Y., Wang, M., Jiang, W.: Autonomous mobile software automation via semantic knowledge graphs and LLM reasoning. *Automated Software Engineering* **33**(4), 126 (2026) <https://doi.org/10.1007/s10515-026-00663-4>
 - [27] Liu, F., Liu, Y., Shi, L., Yang, Z., Zhang, L., Lian, X., Li, Z., Ma, Y.: Beyond functional correctness: Exploring hallucinations in LLM-generated code. *IEEE Transactions on Software Engineering* **52**(3), 1037–1055 (2026) <https://doi.org/10.1109/tse.2026.3657432>
 - [28] Advani, L.: From confident closing to silent failure: Characterizing false success in LLM agents. *arXiv preprint arXiv:2606.09863* (2026) [arXiv:2606.09863](https://arxiv.org/abs/2606.09863)
 - [29] Liu, J., Xia, C.S., Wang, Y., Zhang, L.: Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2023)
 - [30] Salva, S.: Reliable execution of natural language test cases for GUI applications using LLM agents. *Software Quality Journal* **34**(3), 33 (2026) <https://doi.org/10.1007/s11219-026-09767-2>
 - [31] Potdar, A., Shihab, E.: An exploratory study on self-admitted technical debt. In: *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 91–100 (2014). <https://doi.org/10.1109/ICSME.2014.31>
 - [32] Kadavath, S., Conerly, T., Askell, A., et al.: Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221* (2022) [arXiv:2207.05221](https://arxiv.org/abs/2207.05221)
 - [33] Kamath, A., Jia, R., Liang, P.: Selective question answering under domain shift. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5684–5696 (2020). <https://doi.org/10.18653/v1/2020.acl-main.503>
 - [34] Chacón Sartori, C.: The specification gap: Coordination failure under partial knowledge in code agents. *arXiv preprint arXiv:2603.24284* (2026) [arXiv:2603.24284](https://arxiv.org/abs/2603.24284)

- [35] Chacón Sartori, C.: Architectures of error: A philosophical inquiry into human and AI code. *Philosophy & Technology* **39**(1) (2026) <https://doi.org/10.1007/s13347-026-01056-x>
- [36] Ernst, N.A., Baldassarre, M.T.: Registered reports in software engineering. *Empirical Software Engineering* **28**(2) (2023) <https://doi.org/10.1007/s10664-022-10277-5>
- [37] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Experimentation in Software Engineering*. Springer, Berlin, Heidelberg (2012). <https://doi.org/10.1007/978-3-642-29044-2>
- [38] Cameron, A.C., Gelbach, J.B., Miller, D.L.: Bootstrap-based improvements for inference with clustered errors. *Review of Economics and Statistics* **90**(3), 414–427 (2008) <https://doi.org/10.1162/rest.90.3.414>

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- [supplementaryasej.pdf](#)