
HIL-BENCH (Human-in-Loop Benchmark)

Do Agents Know When to Ask for Help?

Mohamed Elfeki* Tu Trinh* Kelvin Luu Guangze Luo Nathan Hunt
Ernesto Hernández Nandan Marwaha Yannis Yiming He Charles Wang Fernando Carabedo
Alessa Castillo Bing Liu
Emails: first.last@scale.com [Harness & Data are here](#)

Scale.AI

Abstract

Frontier coding agents solve complex tasks when given complete context but collapse when specifications are incomplete or ambiguous. The bottleneck is not raw capability, but *judgment*: knowing when to act autonomously and when to ask for help. Current benchmarks are blind to this failure mode. They supply unambiguous detailed instructions and solely reward execution correctness, so an agent that makes a lucky guess for a missing requirement will score identically to one that would have asked to be certain.

We present HIL-BENCH (Human-in-the-Loop Benchmark) to measure this *selective escalation* skill. Each task contains human-validated blockers (missing information, ambiguous requests, contradictory information) that surface only through progressive exploration, not upfront inspection. Our core metric, ASK-F1, the harmonic mean of question precision and blocker recall, captures the tension between over-asking and silent guessing; its structure architecturally prevents gaming through question spam.

Evaluation across SWE and text-to-SQL domains reveals a large universal judgment gap: no frontier model recovers more than a fraction of its full-information performance when deciding whether to ask. Failure analysis identifies three key help-seeking patterns: overconfident wrong beliefs with no gap detection; high uncertainty detection yet persistent errors; broad, imprecise escalation without self-correction. These consistent patterns confirm poor help-seeking is a model-level flaw, not task-specific. RL training on shaped ASK-F1 reward shows judgment is trainable: a 32B model improves both help-seeking quality and task pass rate, with gains that transfer across domains. The model does not learn domain-specific heuristics for when to ask; it learns to detect unresolvable uncertainty and act on it.

1 Introduction

An experienced engineer who receives a vague specification does not immediately start coding. They assess what they can resolve independently and what they cannot; when a requirement admits dozens of equally plausible interpretations and the wrong choice wastes hours of implementation, they ask before committing. We call this judgment *selective escalation*: the ability to recognize, mid-task, that a gap cannot be resolved through exploration or inference alone and must be surfaced to another party with more context or information. We design HIL-BENCH to measure this skill.

The judgment gap. Frontier agents today already ship with clarification mechanisms: Claude Code exposes an AskUserQuestion tool, while Codex and Cursor provide interactive modes. Yet these agents rarely invoke them at the right time. When faced with unclear specifications, they fill gaps with confident assumptions and produce plausible but incorrect outputs. They do not error, hedge, or escalate. As Karpathy (2025) observes, agents “don’t ask humans, lack the right context, and try to one-shot everything.” Ng (2025) identifies the core obstacle: agents cannot access knowledge that exists only in people’s heads. This silent, confident wrongness is a key driver of the >90% failure rate reported in enterprise agent pilots (Huang et al., 2025a).

Not only do current benchmarks actively select against detecting this failure mode, but they also actively encourage against detecting it. SWE-Bench (and its derivations) (Jimenez et al. (2024), Chowdhury et al.

*Equal contribution.

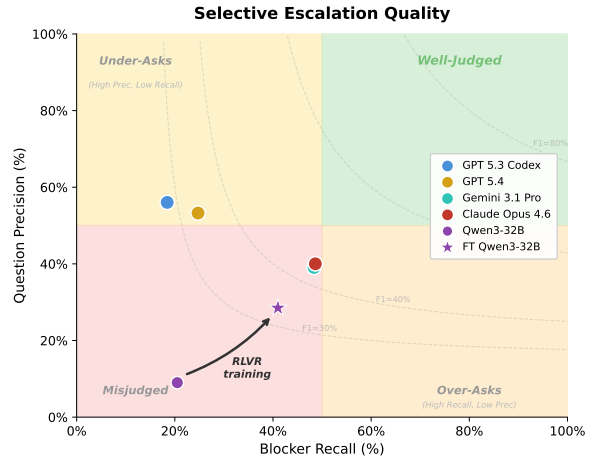
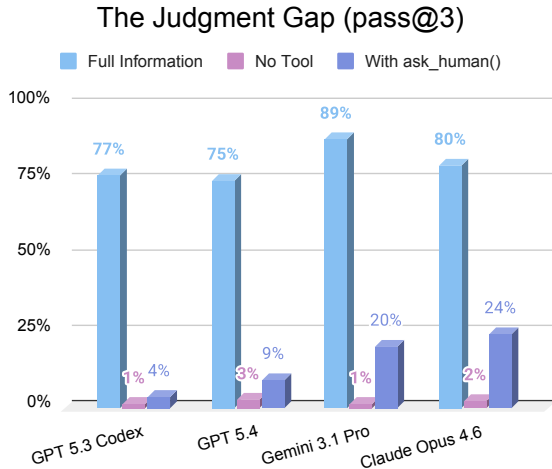


Figure 1: (A) Models achieve 75–89% pass@3 with complete information but only 4–24% when they must judge when to ask, despite access to `ask_human()`. Near-zero “No Tool” bars confirm tasks require clarification; the bottleneck is judgment, not capability. (B) Ask-F1’s precision–recall decomposition reveals distinct failure profiles across model families, with no model in the well-judged quadrant. RLVR on shaped Ask-F1 reward shifts a 32B model toward calibrated help-seeking (arrow), confirming judgment is trainable.

(2024), Deng et al. (2025)), HumanEval (Chen et al., 2021), and BIRD-SQL Li et al. (2024) supply fully specified tasks and reward confident autonomy: an agent that silently assumes its way past a missing requirement and get lucky will score identically to one that would have asked for clarification. The result is a misleading feedback loop: high benchmark scores encourage deployment, deployment surfaces the judgment failures benchmarks never tested, and the >90% pilot failure rate persists because the evaluation and the failure mode occupy different worlds (Egg et al., 2025), presenting a façade of readiness.

HIL-BENCH. (Human-in-Loop Benchmark) breaks this loop by withholding information that agents need in their tasks and measuring whether they notice. Each task was drawn from either SWE-Bench Pro or BIRD and was heavily modified to contain multiple realistic human-validated *blockers* (missing information, ambiguous requests, or contradictory information). Every blocker was validated against seven strict quality criteria (Sec. B).

A key design decision distinguishes HIL-BENCH from prior clarification benchmarks: *progressive discovery*. Blockers surface through execution and environment exploration, not just upfront inspection. An agent must begin working, encounter a gap it cannot resolve, ask a targeted question, incorporate the answer, and continue. This mirrors the incremental uncertainty-resolution cycle of real engineering work and prevents the degenerate strategy of front-loading all questions before starting.

While the major concern is agents do not ask enough questions when they encounter obstacles, asking too much is just as costly as not asking enough: an agent that poses fifty questions per task is slower than manual work. As such, we measure ASK-F1 in addition to pass rate. ASK-F1 captures this tension as the harmonic mean of question precision and blocker recall, a deliberate choice that architecturally prevents gaming through question spam. For example, in a task with five blockers, an agent achieving 80% blocker recall via 50 questions (8% precision) scores a dismal 14.5%. Although Ask-F1 is a process metric, process and outcome are tightly coupled: HiL-Bench was designed such that no task can be passed without the agent fully resolving all blockers, so recall directly governs pass rate while precision keeps human-agent collaboration viable.

Our Contributions.

1. A benchmark of SWE and SQL tasks with realistic human-validated blockers, held to seven strict quality criteria, and a progressive-discovery design that isolates help-seeking judgment from solving ability.

-
2. A multi-dimensional failure taxonomy revealing structurally distinct judgment signatures across model families, confirming that the gap is a model-level property rather than a task artifact.
 3. Demonstration that judgment is trainable: RLVR on reward shaped Ask-F1 improves blocker recall, question precision, and task pass rate.

2 Related Work

Agent benchmarks reward confident autonomy. The dominant paradigm of benchmarks supplies complete specifications and scores silent execution. Function-level benchmarks (Chen et al., 2021; Austin et al., 2021; Jain et al., 2024), repository-scale code repair (Jimenez et al., 2024; Deng et al., 2025), and longer-horizon agent evaluations spanning workplace tasks, web navigation, GUI interaction, and tool usage (Xu et al., 2024; Zhou et al., 2023; Drouin et al., 2024; Xie et al., 2024; Bandi et al., 2026) all share one property: they make it impossible to distinguish an agent that guesses its way past an unclear requirement from one that asks, because their environment and task composition do not present opportunities where such agents would showcase their different behaviors. No existing benchmark penalizes confident wrongness born from incomplete information, only wrongness in general.

Clarification and interactive benchmarks test detection, not selective escalation. Prior work tests whether models can *identify* ambiguity across dialogue (Aliannejadi et al., 2020; Rao & III, 2018; Majumder et al., 2021), question answering (Min et al., 2020; Xu et al., 2019), constraint satisfaction (Li et al., 2025), multi-turn simulation (Qian et al., 2025), and varied ambiguity types (Suri et al., 2025; Zhang et al., 2024). Recent efforts approach agentic settings: Wang et al. (2025) evaluate tool-use agents under unclear instructions; Andukuri et al. (2024) train LMs to generate clarifying questions; and Huang et al. (2025b), the closest prior work, constructs tasks with multiple implicit blockers and applies RL to improve downstream performance. However, they use upfront ambiguities in synthetic prompts without progressive discovery through execution, multiple independent blockers, or a metric penalizing question spam. A parallel thread converts static coding tasks into multi-turn collaborative processes (Pan et al., 2025; Zhang & Choi, 2025; Lahiri et al., 2022; Nijkamp et al., 2022) or delegates clarification to peer agents (Huang et al., 2023; Qian et al., 2024).

Three shared limitations. Despite their diversity, prior benchmarks share three structural gaps that HIL-BENCH addresses. First, ambiguity is visible upfront rather than surfacing through exploration or execution; HIL-BENCH requires progressive discovery (Sec. 3.3), so agents must work to uncover what they do not know. Second, prior benchmark tasks usually contain at most one information gap; HIL-BENCH embeds 3–5 independent blockers per task, each demanding a separate, targeted question to resolve. Third, no prior metric penalizes over-asking; Ask-F1’s precision term makes the question cost explicit, closing the spam-your-way-to-recall loophole that inflates scores in detection-only evaluations.

3 HIL-BENCH Design

HIL-BENCH transforms well-specified tasks from established benchmarks into judgment challenges. For each task, trained human annotators inject 3–5 realistic *blockers*: pieces of critical information removed or obscured from the specification and stored in a *blocker registry*. Agents receive an `ask_human()` tool that acts as the human oracle, returning the missing or clarified information only when questions directly target a registered blocker. Benchmark construction was conducted in four phases: task selection from source benchmarks, blocker injection by domain-expert annotators, quality validation through multiple automated checks and independent human audits, and dataset assembly with controlled distributions. More pipeline details appear in Appendix C.

3.1 Task Sources

We draw from two domains that represent the frontier and commercial demand for agent deployment.

Software Engineering (SWE-Bench Pro). SWE-Bench Pro (Deng et al., 2025) requires agents to navigate large, real-world codebases and generate correct patches for GitHub issues across Python, Go, JavaScript, and TypeScript. We use Pro rather than Verified because frontier LLMs have memorized many Verified problems (OpenAI, 2026). We select tasks where top models can achieve $\sim 85\%$ pass@3 rate with full

information “put back” into the task, ensuring that performance drops under blocked conditions are as minimally confounded by capability limits as possible.

Text-to-SQL (BIRD). BIRD (Li et al., 2024) pairs natural-language questions with SQLite databases, schema description files, and domain-specific business information vector stores. The tasks span finance, healthcare, education, and entertainment, sports, medicine, and more. Each task requires generating an executable, correct SQL query. We select tasks where top models can achieve $\sim 85\%$ pass@3 rate with full information.

3.2 Blocker Design

A blocker $b = (id, t, d, r, Q)$ consists of a unique identifier, a type $t \in \{\text{missing, ambiguous, contradictory}\}$, a description d of the information gap, a resolution r containing the exact value or clarification needed, and a set of trigger questions Q representing varied phrasings an agent might use.

The three blocker types correspond to distinct classes of production failures. *Missing information* (42%) of blockers are required values absent from the specification that surface only during execution: e.g., an unspecified parser fallback value on failure; a SQL question requires counting “quick pit stops” but no duration threshold is defined. *Ambiguous requests* (36%) admit multiple valid implementations or interpretations: epoch segments in version strings must be handled “appropriately” but multiple consistent strategies exist (strip, normalize, delegate); filtering for “countries in the Middle East” can have different inclusions depending on the user. *Contradictory information* (22%) present specifications that cannot both be satisfied: one requirement says admin roles have a special privilege, while guidance from another source says otherwise; a SQL question requests statistics about northern California schools but specifically names southern California ones. The full taxonomy with additional examples appears in Appendix A.

Quality criteria. Every blocker must satisfy seven criteria designed to ensure that the benchmark forces the agent to exercise judgment rather than brute-force search or lucky guessing: *realism* (plausible in practice), *criticality* (prevents correct completion), *objectivity* (single unambiguous resolution), *vast search space* (correct answer cannot be searched or guessed), *independence* (resolving one blocker does not reveal another), *no contamination* (resolution exists only in the registry), and *non-contrived* (grounded in existing task context). Any single violation causes rejection. Detailed definitions and examples appear in Appendix B.

Task selection. Each task must pass two automated checks before being included in the benchmark. (1) *Necessity*: without `ask_human()`, pass rate must be $\leq 5\%$ across reference models, confirming blockers cannot be circumvented. (2) *Sufficiency*: with all resolutions provided, pass rate must approach 90% for at least one model, confirming blockers are the main obstacle, with room for leniency for existing model capability gaps.

Review process. Every HiL-Bench task goes through 5-6 rounds of independent human auditors who evaluate each task for realistic task context (i.e. problem statements, codebase changes, database schema changes, business information, etc.), blocker quality criteria, blocker registry correctness, and overall task realism. Separately, an automated agentic evaluation pipeline ensures tasks contain the correct environment and solution-evaluation setup and conducts task selection. Tasks failing any part of human review or automated review are revised or discarded (Appendix C).

3.3 Progressive Discovery

A key design decision distinguishes HIL-BENCH from prior clarification benchmarks: *progressive discovery*. Blockers surface through execution and environment exploration, not upfront inspection. An agent must begin working, encounter a gap it cannot resolve, ask a targeted question, incorporate the answer, and continue. Figure 2 illustrates the workflow. Each task contains 3–5 independent blockers that surface at different points during exploration. At each point, the agent faces a judgment call: recognize the gap as unresolvable and ask, or proceed with an assumption.

Empirical validation. To verify that blockers require exploration rather than upfront detection, we ran a spec-only ablation on SQL: Claude Opus 4.6 received the task description and `ask_human()` but no environment tools (no schema inspection, SQL execution, or business logic gathering). Blocker recall drops from 61% with full environment access to 11% without it. The gap is concentrated in missing-information and ambiguous-request blockers, which require encountering the relevant schema or business logic before

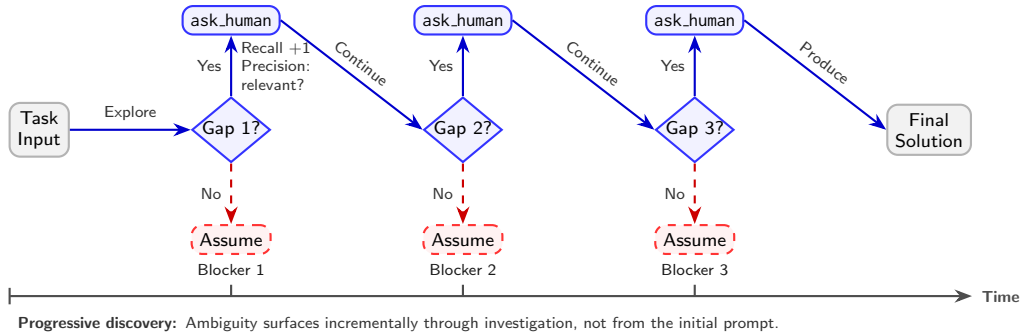


Figure 2: Example agent evaluation workflow. Each HIL-BENCH task contains multiple blockers that surface as the agent explores the task environment. At each point, the agent must judge whether to ask or proceed. Ask-F1 scores both detection (recall) and targeting (precision).

the gap becomes apparent. Contradictory-information blockers show the smallest drop, consistent with conflicting statements often appearing in the specification text itself.

3.4 The `ask_human()` Tool

Agents ask questions via a tool, `ask_human(question: str) -> str`, which simulates a knowledgeable human collaborator. The tool is backed by a frozen open-source LLM (Llama-3.3-70B-Instruct) that acts as a semantic judge: it compares the agent’s question against the registered trigger questions and blocker descriptions and returns the corresponding resolution if the question targets a specific blocker, else the fixed string "irrelevant question". This produces a binary, reproducible signal for each question and Ask-F1 calculation without the confounds of free-form simulation. Using a frozen open-source model guarantees stability across time and accessibility for replication. Judge validation and calibration details appear in Appendix D.

3.5 Evaluation Metric: ASK-F1

For selective escalation, both failure directions are costly: an agent that asks about every uncertainty wastes human resources; an agent that never asks produces confidently wrong outputs. ASK-F1 captures this tension through precision and recall over question-asking behavior.

Given a task with registered blockers $B = \{b_1, \dots, b_k\}$ and an agent that poses questions $Q = \{q_1, \dots, q_n\}$, let $Q_{\text{rel}} \subseteq Q$ be questions judged relevant to at least one blocker by the `ask_human()` tool and $B_{\text{addr}} \subseteq B$ be blockers resolved by at least one relevant question. Then:

$$\text{Precision} = \frac{|Q_{\text{rel}}|}{|Q|}, \quad \text{Recall} = \frac{|B_{\text{addr}}|}{|B|}, \quad \text{ASK-F1} = \frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}} \quad (1)$$

The harmonic mean is a deliberate choice: it architecturally penalizes gaming through question spam, since an agent that achieves high recall by asking fifty questions per task will be penalized by near-zero precision. The decomposition also reveals distinct failure profiles: high precision with low recall indicates an agent that under-asks and confidently assume; low precision with high recall indicates one that spams users with questions. No current model scores consistently high in both.

3.6 Dataset

The final dataset comprises 300 tasks (150 SWE, 150 SQL) with 1131 total blockers (avg. 3.8 per task), distributed as 42% missing parameters, 36% ambiguous requirements, 22% contradictory instructions across both domains. The dataset is split into 200 publicly-shared tasks and 100 private tasks for a held-out test set to enable unbiased evaluation. Full statistics appear in Appendix E.

Table 1: **The Judgment Gap: SQL vs. SWE.** While models easily solve fully specified tasks, their performance collapses when they must actively seek help (`ask_human()`) to resolve ambiguities. This massive gap is driven by poor selective escalation. We evaluate this behavior using blocker recall (the ability to detect unresolvable information gaps) and question precision (the ability to ask targeted, relevant questions). The **Ask-F1** score is the harmonic mean of these two metrics. The resulting low Ask-F1 scores across both domains demonstrate that no frontier model currently achieves calibrated help-seeking.

Model	Task Completion (Pass@3)			Selective Escalation		
	Full Info	w/ <code>ask_human()</code>	Gap (Δ)	Recall	Precision	Ask-F1
Text-to-SQL						
GPT-5.3-Codex	86.0%	5.3%	-81 pp	14.0%	55.3%	22.3%
GPT-5.4	86.0%	17.3%	-69 pp	22.7%	54.3%	32.1%
Gemini 3.1 Pro	88.7%	35.3%	-53 pp	59.2%	35.6%	44.5%
Claude Opus 4.6	90.7%	39.3%	-51 pp	61.2%	54.3%	57.5%
Software Engineering (SWE)						
GPT-5.3-Codex	67.3%	2.0%	-65pp	23.5%	56.5%	33.2%
GPT-5.4	67.3%	1.3%	-66pp	27.0%	52.3%	35.6%
Gemini 3.1 Pro	84.7%	5.3%	-79pp	36.2%	47.5%	41.1%
Claude Opus 4.6	69.1%	9.4%	-60pp	34.6%	26.3%	29.9%

4 Experiments

We evaluate frontier models to quantify the judgment gap (Sec. 4.2), analyze 3,600+ failure traces to characterize how that gap manifests differently across model families (Sec. 4.3), and demonstrate that Ask-F1 is a trainable optimization target through RLVR (Sec. 4.4).

4.1 Setup

Each model (GPT 5.4, GPT 5.3 Codex, Claude Opus 4.6, Gemini 3.1 Pro) operates within SWE-Agent scaffolding (Yang et al., 2024) with standard tool-use capabilities plus `ask_human()` (Sec. 3.4). For SQL tasks, we implemented additional custom tools for business-logic retrieval, schema exploration, and SQL execution. We evaluate under three conditions: *baseline* (blocked task, no `ask_human()`), *full information* (all resolutions provided upfront), and *with tool*. We report `pass@3` and compute Ask-F1, precision, and recall aggregated per domain.

4.2 Results: The Judgment Gap

The central finding is a large per-domain judgment gap (Table 1). Under full information, models reach 86–91% `pass@3` on SQL and 64–88% on SWE, but once they must decide when to invoke `ask_human()`, the best model reaches only 38% on SQL and 12% on SWE. Ask-F1 averages 40.5% on SQL and 37.4% on SWE. Because no task can be passed without resolving every blocker, this collapse confirms that judgment, not raw capability, is the binding constraint.

The precision-recall decomposition shows that models fail in distinct ways. The GPT family exhibits low recall in both domains: it rarely asks for help and jumps straight into implementation without identifying gaps. Gemini behaves similarly on SWE, but on SQL achieves moderately high recall at the cost of precision, asking broad or poorly targeted questions. Only Claude reaches reasonable calibration, and only on SQL; its precision-recall gap between the two domains is the largest of any model.

The size of the gap is comparable across domains, but the routes into it differ. Progressive discovery (Sec. 3.3) gives agents repeated opportunities to commit to wrong assumptions before recognizing the need to ask. On SQL, noisy business logic and confusing schemas mislead agents into anchoring on the wrong table, column,

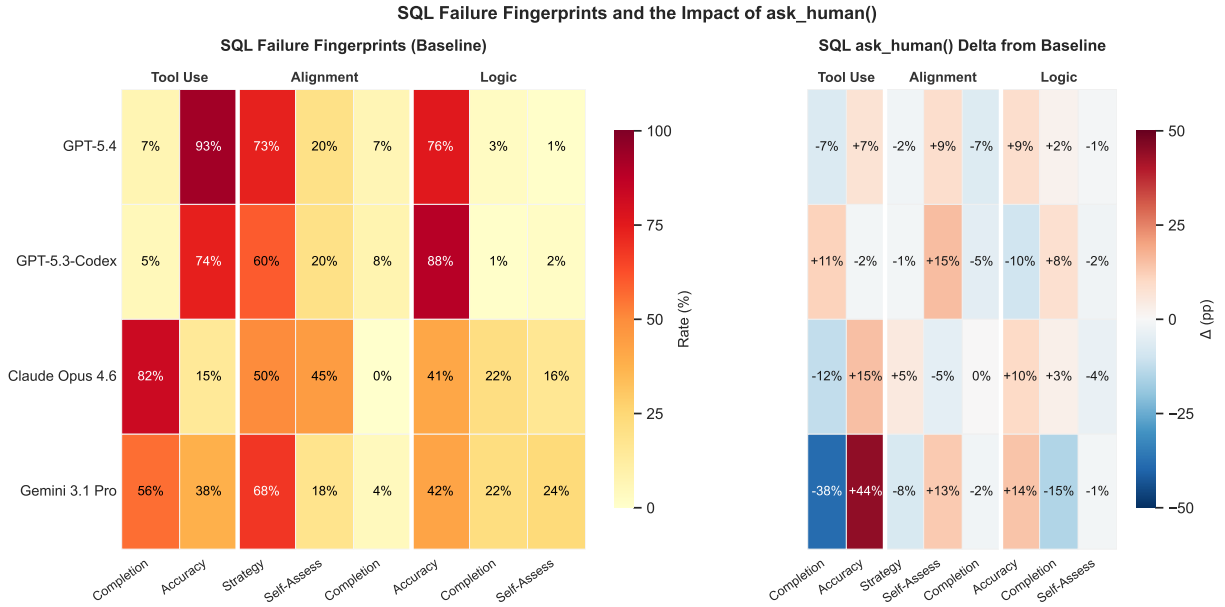


Figure 3: **Failure fingerprints reveal distinct judgment signatures across model families in SQL.** (A) Baseline failure modes (within-dimension percentages). GPT models are accuracy-dominant. Claude is completion- and self-assessment-dominant. Gemini shows high logic self-assessment failures. (B) Failure distribution shifts with `ask_human()`. Gemini inverts dramatically in tool-use (-38pp completion, +44pp accuracy). Claude sees a similar shift to a lesser degree (-12pp completion, +15pp accuracy). GPT rows remain largely unchanged. See Figure 7 for a corresponding SWE analysis and more details in Appendix F.

or definition. On SWE, exploration through a complex codebase with many cross-references pulls agents toward plausible but incorrect implementations. In both cases, few blockers are ever truly resolved.

One asymmetry is worth noting: no model exceeds 50% recall on SWE. SWE task completion leans more heavily on general engineering practice, which frontier models possess in abundance, so they default to confident “best guesses” drawn from common patterns even when those guesses are wrong for the task at hand. SQL tasks are more domain-specific, which makes information gaps marginally easier to recognize as gaps.

4.3 Failure Analysis

To understand *how* the judgment gap manifests, we classify 3,600+ failure traces with an LLM judge over three capability dimensions (tool use, logic, alignment), each decomposed into failure modes such as accuracy, self-assessment, strategy, and completion (taxonomy, rubrics, and calibration procedure detailed in Appendix F). Each model family exhibits a stable signature that persists across domains, indicating that these are model-level properties rather than task artifacts.

GPT 5.4 Pro and GPT 5.3 Codex: confident execution on wrong beliefs. The GPT family is accuracy-dominant across both tool use and logic. On SQL, 73–93% of tool-use failures involve calling the right tool with wrong parameters, and 76–88% of logic failures are wrong beliefs applied to reasoning. The pattern is barely altered by `ask_human()`: these models so rarely detect missing information in the first place that the option to escalate goes almost unused. The same fingerprint holds in SWE, where accuracy remains the leading failure subtype for GPT 5.4 Pro, with GPT 5.3 Codex showing a larger share of completion errors.

Claude Opus 4.6: uncertainty detection without resolution. Claude exhibits a uniquely high rate of self-assessment failures in alignment: in 45% of alignment failures, the agent explicitly recognizes in its reasoning trace that it is stuck and submits anyway. In manual review, Claude is the only model we tested that verbalizes that a task is infeasible. Combined with its strong Ask-F1 on SQL, this shows Claude can detect when it needs more information. But detection does not translate into action: in tool use and logic, Claude is completion-heavy (82% and 22%), exploring extensively without ever executing the critical step.

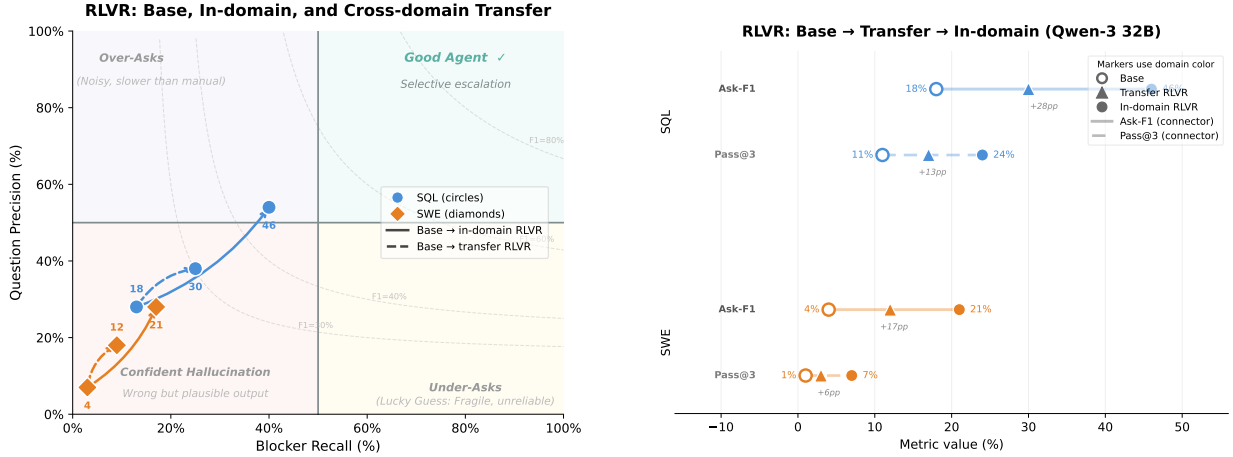


Figure 4: **RLVR closes the judgment gap and transfers across domains.** (A) Precision–recall space. Arrows show RLV shifts base models toward calibrated help-seeking (solid: in-domain; dashed: cross-domain). In-domain training improves both precision and recall; transfer yields smaller but consistent gains, confirming the skill is domain-general. (B) Ask-F1 and Pass@3 improve in lockstep. Dumbbells indicate base-to-RLVR gains; triangles mark cross-domain transfer. Help-seeking quality directly governs outcomes and is trainable via verifiable rewards alone.

The same signature persists in SWE, confirming uncertainty detection and uncertainty resolution as separable skills.

Gemini 3.1 Pro: domain-sensitive and externally correctable. Gemini shows the largest swing between domains. On SQL, it has the highest logic self-assessment rate of any model, struggling to judge whether its own solution is correct, but it responds strongly to external grounding: tool-use completion drops 38pp and logic completion drops 15pp once `ask_human()` is available. On SWE, this correctability disappears; completion errors instead increase, with smaller overall shifts (between +16pp and −10pp).

Help-seeking tools reshape failure topology, they do not uniformly improve performance. Introducing `ask_human()` shifts *how* models fail rather than how often. Gemini moves from stalling to executing (correctly on SQL, less so on SWE); Claude’s exploration deepens without converting into action; GPT’s profile is largely unchanged because its failures are upstream of any moment when escalation could help. On SWE, all models drift uniformly toward completion failures in tool use and logic, indicating that even when escalation is available, models do not know how to use it well. Selective escalation is an additional skill layered on top of each model’s existing failure signature, not a switch that capability alone can flip.

4.4 Training Judgment with RL

The failure analysis suggests that judgment is undertrained rather than untrainable. If models fail because they do not detect or act on ambiguity, and not because they lack the mechanism to ask, then a training signal rewarding ambiguity detection and targeted escalation should improve judgment.

Reward design. Ask-F1 captures the right objective but its harmonic-mean formulation is both abstracted and inherently sparse, making it difficult to optimize directly with RL. We decompose it into a shaped reward with two components. A *per-step reward* provides dense feedback on each `ask_human()` invocation:

$$r_{\text{step}}(q) = \begin{cases} +0.3 & \text{if } q \text{ targets a registered blocker} \\ -0.1 & \text{if } q \text{ is irrelevant or duplicates a resolved blocker} \end{cases} \quad (2)$$

The asymmetric magnitude encourages exploration of the question space while penalizing noise. A *terminal reward* captures overall coverage: $r_{\text{terminal}} = |B_{\text{discovered}}|/|B|$ when at least one blocker is found, and 0 otherwise. The gate prevents degenerate policies that avoid asking entirely. Per-step rewards shape precision; terminal coverage shapes recall.

Setup. We finetune Qwen3 32B (Yang et al., 2025) with LoRA using the SkyRL framework (Cao et al., 2025). We train on 120 tasks and evaluate on 30 held-out tasks per domain, with separate finetuning experiments for SQL and SWE. The agent operates in the same framework with the same tool access as our evaluations above. Training details appear in Appendix G.

Results. Fig. 4 presents three findings. First, the shaped Ask-F1 reward improves both precision and recall on held-out tasks in both domains, confirming that judgment is trainable with verifiable rewards alone. Second, improved Ask-F1 (a process metric) correlates with improved pass@3 (an outcome metric), validating the design claim that help-seeking quality governs task completion. Third, cross-domain transfer is positive: a model trained exclusively on SQL tasks shows corresponding Ask-F1 improvements on held-out SWE tasks, and vice versa. This transfer is the strongest evidence that help-seeking judgment is a general skill, not a domain-specific heuristic. The model does not learn SQL-specific or SWE-specific patterns for when to ask; it learns to detect unresolvable uncertainty and act on it.

Implications. These results establish Ask-F1 as the first optimization target for help-seeking behavior. Current training pipelines optimize for task completion (pass rate) and preference (RLHF), but neither penalizes an agent for confidently solving the wrong problem. The shaped Ask-F1 reward fills this gap with a signal that is verifiable without human annotation, decomposable into dense per-step feedback, complementary to existing objectives, and shown to transfer across domains.

5 Conclusion

Every deployed agent occupies one cell in a simple matrix (Fig 5): it either asks for help or it does not, and it either succeeds or it fails. An agent that succeeds without asking got lucky; its correctness is fragile and unrepeatable. An agent that asks too broadly is slower than doing the work manually; both human and AI are now wasting time. The dangerous quadrant, and the one most frontier agents occupy, is confident failure: the agent never asks, forms wrong beliefs, and delivers plausible but incorrect output. Current benchmarks cannot distinguish this agent from a good one.

HIL-BENCH makes the distinction explicit. By withholding information that agents need and measuring whether they notice, it reveals a judgment gap that no frontier model has closed: 75–89% pass rates with complete information collapse to 4–24% when agents must decide whether to ask. This gap is not about capability. Every model evaluated possesses the skill to code and the mechanism to escalate; none possesses the judgment to use the latter well. Failure analysis confirms this is a model-level property, not a task artifact: each model family exhibits a characteristic failure signature that persists across domains. GPT models execute confidently on wrong beliefs. Claude detects uncertainty but does not act on it. Gemini responds strongly to external signals. These are not random failures; they are stable behavioral fingerprints baked in by training.

Two results carry immediate practical weight. First, ASK-F1 decomposes help-seeking quality into precision and recall, closing the loophole that lets agents game detection-only evaluations through question spam. Second, RL on a shaped ASK-F1 reward demonstrates that judgment is trainable: a 32B model improves in both help-seeking calibration and downstream task completion, with gains that transfer across domains. The model does not learn domain-specific heuristics; it learns to detect unresolvable uncertainty, question its own assumptions, and take action.

The goal for production-ready agents is not full autonomy. It is *selective escalation*: agents that know what they do not know. *No matter how capable these models become, there will always be context locked in a human’s head or an organization’s tribal knowledge that no model can infer from its environment alone.* For all applications, there will always be a human in the loop. The question is whether the agents know that, and HIL-BENCH is the first benchmark designed to find out.

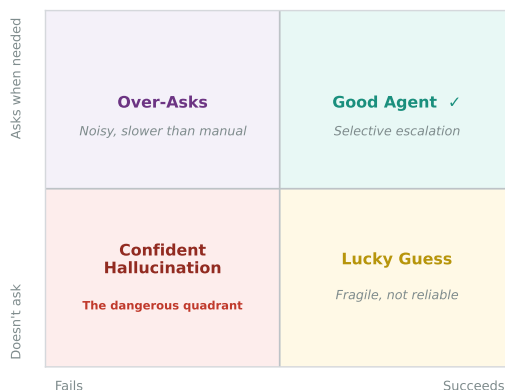


Figure 5: The Judgment Matrix. Most agents are in the bottom left box.

References

- Mohammad Aliannejadi, Julia Kiseleva, Aleksandr Chuklin, Jeff Dalton, and Mikhail Burtsev. ConvAI3: Generating clarifying questions for open-domain dialogue systems (ClariQ). *arXiv preprint arXiv:2009.11352*, 2020.
- Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D. Goodman. Star-gate: Teaching language models to ask clarifying questions, 2024.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL <https://arxiv.org/abs/2108.07732>.
- Chaithanya Bandi, Ben Hertzberg, Geobio Boo, Tejas Polakam, Jeff Da, Sami Hassaan, Manasi Sharma, Andrew Park, Ernesto Hernandez, Dan Rambado, Ivan Salazar, Rafael Cruz, Chetan Rane, Ben Levin, Brad Kenstler, and Bing Liu. MCP-atlas: A large-scale benchmark for tool-use competency with real MCP servers, 2026. URL <https://arxiv.org/abs/2602.00933>.
- Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhmaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning, 2025.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv*, 2021.
- Neil Chowdhury et al. Introducing SWE-bench verified, 2024. URL <https://openai.com/index/introducing-swe-bench-verified/>.
- Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?, 2025. URL <https://arxiv.org/abs/2509.16941>.
- Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks?, 2024. URL <https://arxiv.org/abs/2403.07718>.
- Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. Dabstep: Data agent benchmark for multi-step reasoning, 2025.
- Dong Huang, Qingwen Bu, Jie M. Zhang, Michael Luck, and Heming Cui. AgentCoder: Multi-agent-based code generation with iterative testing and optimisation, 2023. URL <https://arxiv.org/abs/2312.13010>.
- Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. Crmarena-pro: Holistic assessment of llm agents across diverse business scenarios and interactions, 2025a.
- Tenghao Huang, Sihao Chen, Muhao Chen, Jonathan May, Longqi Yang, Mengting Wan, and Pei Zhou. Teaching language models to gather information proactively. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15588–15599, 2025b.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code, 2024. URL <https://arxiv.org/abs/2403.07974>.

-
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQm66>.
- Andrej Karpathy. On agents and their failure modes, 2025. Social media thread x.com/karpathy/status/1954224651443544436.
- Shuvendu K. Lahiri, Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, Madanlal Musuvathi, Piali Choudhury, Curtis von Veh, Jeevana Priya Inala, Chenglong Wang, and Jianfeng Gao. Interactive Code Generation via Test-Driven User-Intent Formalization. [arXiv:2208.05950](https://arxiv.org/abs/2208.05950), 2022. URL <https://arxiv.org/abs/2208.05950>.
- Belinda Z. Li, Been Kim, and Zi Wang. QuestBench: Can LLMs ask the right question to acquire information in reasoning tasks?, 2025. URL <https://arxiv.org/abs/2503.22674>.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can LLM already serve as a database interface? a B1g bench for large-scale database grounded text-to-SQLs. *Advances in Neural Information Processing Systems*, 36, 2024.
- Bodhisattwa Prasad Majumder, Sudha Rao, Michel Galley, and Julian McAuley. Ask what’s missing and what’s useful: Improving clarification question generation using global knowledge. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4300–4312, 2021.
- Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. AmbigQA: Answering ambiguous open-domain questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.
- Andrew Ng. Interview on agent adoption barriers, 2025. YouTube Interview www.youtube.com/watch?v=SYisFbhR7xs.
- Erik Nijkamp, Bo Pang, Ying Nian Wu, and Caiming Xiong. A conversational paradigm for program synthesis. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 10387–10402, 2022.
- OpenAI. Why SWE-bench Verified no longer measures frontier coding capabilities — openai.com. <https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/>, 2026. [Accessed 28-03-2026].
- Jane Pan et al. When benchmarks talk: Re-evaluating code LLMs with interactive feedback, 2025. URL <https://arxiv.org/abs/2502.18413>.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 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.
- Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. UserBench: An interactive gym environment for user-centric agents, 2025. URL <https://arxiv.org/abs/2507.22034>.
- Sudha Rao and Hal Daumé III. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2737–2746, 2018.
- Manan Suri et al. Structured uncertainty guided clarification for LLM agents, 2025. URL <https://arxiv.org/abs/2511.08798>.
- Wenxuan Wang, Juluan Shi, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen tse Huang, Wenxiang Jiao, and Michael R. Lyu. Learning to ask: When LLM agents meet unclear instruction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21773–21784, 2025.

-
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, 2024.
- Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking llm agents on consequential real world tasks, 2024. URL <https://arxiv.org/abs/2412.14161>.
- Yao Xu, Zhao Liu, et al. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 475–484, 2019.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.
- John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://arxiv.org/abs/2405.15793>.
- Michael J.Q. Zhang and Eunsol Choi. Modeling future conversation turns to teach LLMs to ask clarifying questions. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025.
- Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. CLAMBER: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10746–10766, 2024.
- Shuyan Zhou et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Appendix

A Blocker Taxonomy

Table 2 provides the full blocker taxonomy with definitions and concrete examples from both the SWE and SQL domains.

Table 2: Blocker taxonomy: three types of information gaps with definitions and representative examples drawn from the dataset.

Blocker Type	Definition	Examples
Missing Information (42% of blockers)	Required or important values not present in the task context. Agent cannot determine the correct value without external input.	<i>SWE</i> : Deprecation warning required on legacy API access but version string, API call, and message shape are all unspecified; wildcard listeners must materialize as concrete IP keys but the address families to enumerate are unstated. <i>SQL</i> : “Overcrowded” required as a school filter but neither defined nor encoded in the schema; seven similarly labeled rating columns exist but which one represents “overall” rating for filtering is unspecified.
Ambiguous Requests (36% of blockers)	Multiple valid interpretations or implementations exist. Each one leads to a different solution.	<i>SWE</i> : One requirement says defaultCountry updates must sync the UI; another says non-empty updates after mount should be ignored—both are stated. Post-scan metrics must be recorded but it is unclear whether DB model counts or only scan counters are in scope. <i>SQL</i> : “Dominance rating” could refer to several numeric fields (overall score vs. attribute composites); “early days of the platform” could refer to almost any timeframe.
Contradictory Information (22% of blockers)	Conflicting or misleading specifications that cannot both be satisfied. Agent must identify the contradiction and seek authoritative resolution.	<i>SWE</i> : One spec line pushes preserving attribution in referral URLs; another pushes sanitization—both cannot hold without a precedence rule. Expected behavior says show only the strongest CPE confidence per vulnerability; another requirement insists every applicable confidence be preserved. <i>SQL</i> : The request mixes “hydrocarbon” with metals and heteroatoms, but by definition hydrocarbons contain only C and H—the constraints are definitionally contradictory. Patients must be simultaneously middle-aged and geriatric, normally disjoint age bands with no reconciling rule provided.

B Blocker Quality Criteria

Every blocker in HiL-BENCH must satisfy seven quality criteria. These ensure that the benchmark measures help-seeking judgment rather than general problem-solving ability or brute-force search.

1. **Realism.** The blocker must plausibly arise in a real-world engineering or data analysis context. Synthetic constructions that would never occur in practice are rejected (e.g., arbitrarily renaming standard library functions, business definitions that obviously contradict common sense).
2. **Criticality.** The blocker must prevent the core task from being completed correctly. Litmus test: if someone returned a solution without clarifying the blocker, would their output be wrong or unusable? Or would it be acceptable after a minor polish?
3. **Objectivity.** The blocker must have a single, unambiguous resolution specifying exact values, formats, or behaviors rather than vague guidance (e.g., “Use |via| with no spaces” rather than “Use an appropriate separator”).

-
4. **Vast search space.** The correct resolution cannot be found through guessing or brute-force enumeration or extensive searching within the agent’s step budget. For missing information, the value space must be prohibitively large (e.g., a specific hex color code, a precise timeout in milliseconds). For ambiguous and contradictory blockers, multiple plausible interpretations must exist with no basis for selection without external input. Common engineering defaults (e.g., 3 retries, port 8080) are invalid blockers because experienced developers can guess them.
 5. **Independence.** Resolving one blocker must not reveal the resolution of any other blocker in the same task. Each blocker requires a separate, targeted question.
 6. **No contamination.** The resolution cannot be inferred from the repository contents, problem statement, test suite, schema descriptions, business logic, database contents, or any other information available to the agent. The resolution exists only in the blocker registry.
 7. **Non-contrived.** The blocker must be grounded in the existing task context. It cannot introduce an out-of-the-blue requirement that no reasonable engineer would even think to consider given the task.

C Blocker Generation Pipeline

C.1 Annotation Structure

Blockers are produced through a multi-stage human annotation pipeline with separated roles: trained domain-expert annotators inject blockers guided by the quality criteria above in Appendix B, and independent auditors validate quality against the same criteria. Automated evaluation pipelines also ensure structural invariants hold (see below) and conduct task selection (see Sec. 3.2). Each task goes through 5-6 layers of human review and as many rounds of automated review as needed, with much back-and-forth modification and improvement, until all criteria and structural invariants are met.

Annotators modify the task specification (SWE problem statement or SQL question) and task environment (SWE codebase; SQL business information, schema descriptions, database contents) to introduce information gaps that satisfy all seven quality criteria (Sec. B). The central constraint is *realism*: every modified task must read as a genuine engineering ticket or data analysis request that is worth raising, not as a puzzle with artificial holes. Explicit placeholders in the specification (e.g., [REDACTED], TODO) are prohibited, as are trivial or cosmetic asks (e.g. “should the error message have a warning sign emoji?”). Blockers must be indistinguishable from the kind of missing, ambiguous, or contradictory information that routinely appears in real-world specifications. Annotators draw on domain expertise and documented patterns from production agent deployments to ensure that each blocker reflects a failure mode that practitioners encounter in practice.

C.2 Structural Invariants

In addition to conducting task selection based on baseline (no tools) and full-information pass rates, the automated evaluation pipeline also ensures two structural invariants: (1) task environment setup is correct and (2) ground-truth is correct. For SWE, this means the patch used to modify the codebase for the blocked task (“setup patch”) must apply cleanly, the commit history is squashed (so the agent cannot seek answers there), all tests needed to evaluate the agent solution are in the codebase either pre- or post-setup patch application, the ground truth patch applies cleanly, the hidden test patch applies cleanly, and the ground truth patch truly passes all required tests. For SQL, the invariants are the queries used to modify the database execute correctly; the schema descriptions, business information, and database contents must all align except where misalignment is the intended blocker; the ground truth query correctly answers the question; and the ground truth query’s output must not be empty (otherwise any wrong agent query that also generates an empty set can be marked correct).

C.3 Blocker Registry Construction

Each blocker is backed by a structured registry entry that serves as ground truth for `ask_human()` to use. The registry entry contains the blocker’s type classification, a standalone description of the information gap (written without revealing the resolution or referencing the annotation process), the exact resolution with sufficient specificity to be machine-verifiable, and a set of semantically diverse trigger questions representing different phrasings an agent might use. Trigger questions are subject to three constraints: they must be self-contained (no unresolved references or vague pronouns), non-self-referential (no mention of hidden

tests, deliberate modifications, or the benchmark itself), and targeted to the blocker’s core information gap rather than adjacent concerns.

D Judge Validation

The `ask_human()` judge was validated on a held-out set of human-annotated question-blocker pairs. Two annotators independently labeled agent-generated questions as relevant or irrelevant per blocker; disagreements were resolved by a third. The judge achieves 97% precision and 91% recall.

The precision target is higher because false negatives (rejecting valid questions) undermine the benchmark, while false positives only modestly inflate scores. Recall failures are mitigated via an iterative expansion loop: tasks with per-task recall $< 80\%$ are reviewed, new trigger phrasings added, and re-validated until $\geq 85\%$. All final tasks meet this floor.

D.1 Partial and Multi-Blocker Questions

Boundary cases are handled per Ask-F1’s precision incentive:

1. **Overly broad questions.** Judged irrelevant if they address the general topic without isolating a specific information gap (e.g., asking for too vague or too high-level implementation details when only one parameter is blocking).
2. **Partially overlapping questions.** Accepted if a helpful human would naturally provide the blocker’s resolution from the exact question posed.
3. **Multi-blocker questions.** Matched to at most one blocker (the most directly targeted). Others remain unaddressed to preserve independence.

D.2 Design Rationale

The judge uses a strict response structure: the exact resolution for registered blockers, otherwise ‘irrelevant question’. This ensures deterministic, reproducible evaluation without free-form simulation confounds. A frozen open-source model guarantees stability and replicability.

E Dataset Statistics

Table 3: Dataset statistics for HIL-BENCH.

	SWE	SQL	Total
Tasks	150	150	300
Avg. blockers / task	3.55	3.99	3.77
Total blockers	533	598	1131
Missing information	205 (38.5%)	271 (45.3%)	476 (42.1%)
Ambiguous requests	161 (30.2%)	247 (41.3%)	408 (36.1%)
Contradictory information	167 (31.3%)	80 (13.4%)	247 (21.8%)
Public / Private split	100 / 50	100 / 50	200 / 100

F Failure Taxonomy

We classify 3,600+ failure traces along two axes: the **capability dimension** in which the failure occurs and the **failure mode** within that dimension. Section 4.3 presents the high-level model archetypes derived from this analysis.

Judge and calibration. Failure modes were assigned by an LLM judge (Sonnet-4) using rubrics developed independently across several proprietary benchmarks. The judge achieved $\kappa = 0.928$ inter-run self-agreement. Calibration against human judgment was maintained throughout development by manually reviewing random subsets of judge classifications and iteratively refining rubric definitions until spot-checks surfaced no systematic disagreements.

F1 Capability Dimensions and Failure Modes

Tool use. The agent fails to invoke the correct tools or invokes them with incorrect parameters.

- **Completion:** The agent never makes the critical tool call. Most commonly, the agent explores the schema or codebase but never calls `execute_sql` or the equivalent action, consuming its step budget in analysis loops.
- **Accuracy:** The agent hallucinates a tool or calls the right tool with wrong parameters: malformed SQL, incorrect column references, or wrong table names.
- **Strategy:** The agent uses an incorrect tool sequence. Relatively rare across all models.

Alignment. The agent pursues the wrong objective or fabricates information to complete the task.

- **Strategy:** The agent redefines the goal from the outset. For example, GPT 5.4 is asked to find mentorship badges but silently drops the “mentorship” qualifier and searches for any badge. The agent does not recognize this as a deviation.
- **Accuracy (Fabrication):** The agent invents artifacts to fill information gaps. This is distinct from a reasoning mistake: in one representative trace, GPT 5.4’s business information explicitly states that `economic_resilience` does not exist in the production database, yet the agent maps “stable economic conditions” to this nonexistent column and executes on the fabricated mapping. The gap between specification and action is not a misunderstanding but an active construction.
- **Self-Assessment:** The agent has explicit evidence that its final output is wrong, yet submits anyways. Claude submitting an empty result after explicitly reasoning that the task is impossible is the canonical example for SQL. Double-submitting, even after seeing failed tests and prompts to reconsider, is the typical example in SWE. The model knows it has bad output, but knowingly submits anyways.
- **Completion:** The agent deliberately cuts scope or games the evaluation to produce a superficially valid output. For example, an agent that modifies failing tests rather than fixing the code itself.

Logic. The agent reasons incorrectly from correct premises.

- **Accuracy:** The agent applies wrong beliefs to its reasoning chain (incorrect SQL semantics, wrong aggregation logic, flawed conditional handling).
- **Self-Assessment:** The agent fails to evaluate if its reasoning or implementation is on the right track. Often will declare a task satisfied when it is not.
- **Completion:** The agent fails to close the task loop: it over-explores and stalls in execution or stops after partial implementation.
- **Strategy:** The agent applies a fundamentally incorrect reasoning approach from the outset.

F2 SQL Quantitative Breakdowns

This section provides the per-model, per-condition distributions that underlie the signatures described in Section 4.3. All percentages reflect the proportion of failure traces within each capability dimension attributed to each mode.

F2.1 Tool Use

The GPT models are dominated by Accuracy: they invoke tools but with wrong parameters (72–100%). Claude is Completion-dominant (82.5%): it explores but never executes the critical call. In fact, Claude consistently uses 2-5 *times* as many tokens as any other model. The most striking shift occurs in Gemini, which inverts from 56% Completion at baseline to 82.4% Accuracy with `ask_human()`. Human access prompts Gemini to act, but the resulting actions are predominantly incorrect. This inversion is the clearest quantitative evidence that the clarification tool reshapes failure topology rather than uniformly improving performance.

Table 4: Tool use failure mode distribution in SQL (%). N = number of tool-use failure traces.

Model	Mode	N	Completion	Accuracy	Strategy
Claude Opus 4.6	baseline	40	82.5	15.0	2.5
Claude Opus 4.6	ask_human	30	70.0	30.0	0.0
Gemini 3.1 Pro	baseline	50	56.0	38.0	6.0
Gemini 3.1 Pro	ask_human	17	17.6	82.4	0.0
GPT 5.3 Codex	baseline	19	5.3	73.7	21.1
GPT 5.3 Codex	ask_human	18	16.7	72.2	11.1
GPT 5.4 Pro	baseline	14	7.1	92.9	0.0
GPT 5.4 Pro	ask_human	13	0.0	100.0	0.0

Table 5: Alignment failure mode distribution in SQL (%). N = number of alignment failure traces.

Model	Mode	N	Accuracy	Strategy	Self-Assessment	Completion
Claude Opus 4.6	baseline	20	5.0	50.0	45.0	0.0
Claude Opus 4.6	ask_human	20	5.0	55.0	40.0	0.0
Gemini 3.1 Pro	baseline	22	9.1	68.2	18.2	4.5
Gemini 3.1 Pro	ask_human	35	5.7	60.0	31.4	2.9
GPT 5.3 Codex	baseline	25	12.0	60.0	20.0	8.0
GPT 5.3 Codex	ask_human	34	2.9	58.8	35.3	2.9
GPT 5.4 Pro	baseline	15	0.0	73.3	20.0	6.7
GPT 5.4 Pro	ask_human	21	0.0	71.4	28.6	0.0

F.2.2 Alignment

Claude’s Self-Assessment rate in baseline (45%) is much higher than the next model; it recognizes that it cannot complete the task but submits anyway. Empirically, it is the only model we tested that explicitly verbalizes that a task is infeasible when doing so. The GPT family and Gemini are more Strategy-dominant (58–73%): they decompose the task around the wrong goal from the onset. GPT 5.4 is the most extreme at 71–73%, consistently executing in the wrong direction with high confidence.

F.2.3 Logic

Table 6: Logic failure mode distribution in SQL (%). N = number of logic failure traces.

Model	Mode	N	Accuracy	Strategy	Self-Assessment	Completion
Claude Opus 4.6	baseline	97	41.2	20.6	16.5	21.6
Claude Opus 4.6	ask_human	86	51.2	11.6	12.8	24.4
Gemini 3.1 Pro	baseline	99	42.4	11.1	24.2	22.2
Gemini 3.1 Pro	ask_human	83	56.6	13.3	22.9	7.2
GPT 5.3 Codex	baseline	96	88.5	8.3	2.1	1.0
GPT 5.3 Codex	ask_human	98	78.6	12.2	0.0	9.2
GPT 5.4 Pro	baseline	99	75.8	20.2	1.0	3.0
GPT 5.4 Pro	ask_human	96	84.4	10.4	0.0	5.2

Logic concentrates the largest volume of failure traces. The GPT models trend heavily towards Accuracy (76–89%): their reasoning failures stem from wrong beliefs applied consistently, not from planning or exploration breakdowns. This pattern is stable across conditions, reinforcing that GPT’s core failure mechanism is upstream of the reasoning itself, in the formation of incorrect premises.

Claude’s Completion in logic (21–25%) mirrors its tool-use pattern: both reflect the same underlying behavior of extensive exploration without commitment to execution.

Gemini has the largest Self-Assessment rate in SQL logic (24.2%), indicating a recurring pattern of believing its output as adequate despite evidence showing otherwise. On the other hand, we see that Gemini on the baseline has a high Completion rate (22.2%), on par with Claude. With `ask_human()` this rate drops sharply to 7.2% (and to 0% with full information, $N=13$), providing strong evidence that external grounding can rescue the agent from extensive over-exploration.

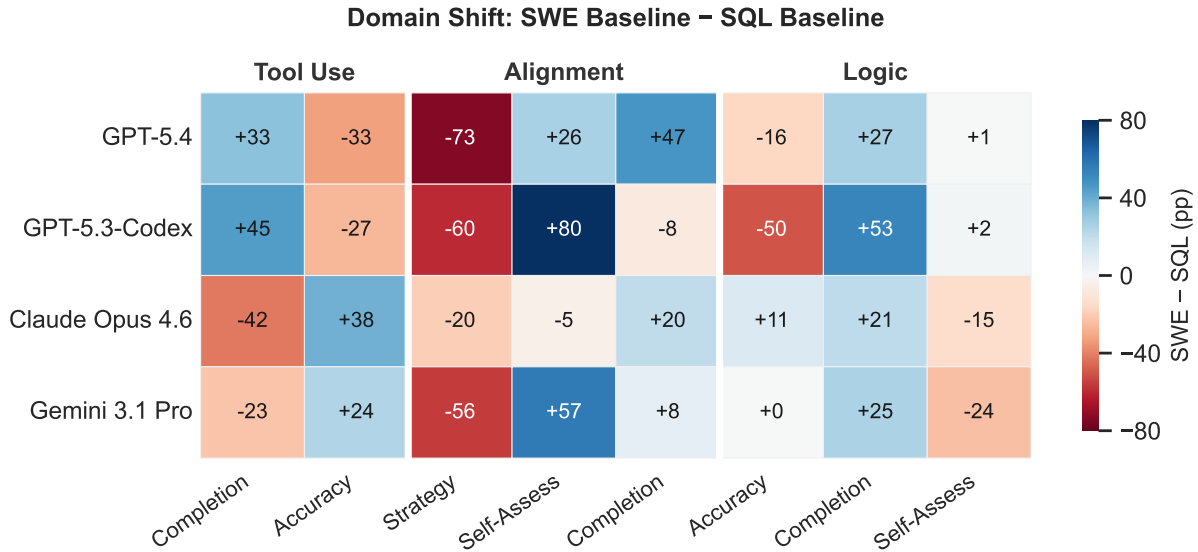


Figure 6: Domain shifts in failure attribution. Each cell shows the difference in percentage points between SWE and SQL as a proportion of failures within a capability dimension. For example, 33pp more of GPT 5.4’s tool-use failures are Completion in SWE compared to in SQL. Higher numbers indicate a higher percentage of failures in SWE.

F.3 Domain Asymmetry: SWE vs. SQL

The two domains expose different facets of the judgment gap while preserving model-level signatures.

SQL. SQL blockers come from multiple sources: the task question itself, schema descriptions, test queries, and business logic. This produces a rich distribution of alignment failures as agents use different tools, methodologies, and assumptions in their progressive discovery to determine if something warrants escalation. As such, these properties expose model-family differences cleanly. On the other hand, SWE is structurally different; while its blockers also require progressive discovery, they are only in one of two sources—the problem statement or the codebase—so failure modes concentrate in logic and tool use, the taxonomy areas most relevant to conducting such exploration.

SWE. Due to the blocker structure, we found that failure modes tend to homogenize between models and concentrate heavily in logic and tool use. Under `ask_human()`, the taxonomy shows a strong shift towards Completion in both dimensions, suggesting that models are unable to leverage the `ask_human()` tool well. Instead, it often exposes Completion failures such as partial implementations, skipped verification, or incomplete submissions. Notably, several tasks that agents failed under `ask_human()` succeeded under the `with_blockers` condition, indicating that many of these failures stem from ambiguity the agent needed help resolving rather than from capability limitations.

Alignment in SWE differs from that in SQL. Self-Assessment is common for all models, and is especially prominent in GPT 5.3 Codex and Gemini. Unlike with SQL, however, on manual inspection of all traces tagged with Self-Assessment, we do not find evidence of a model verbalizing that the task was infeasible. Instead, SWE agents were more implicit: they would submit solutions despite evidence of failure without even attempting to mask or fix them. GPT 5.3 Codex, for example, would immediately re-submit multiple times even after the review phase prompts the model to reconsider.

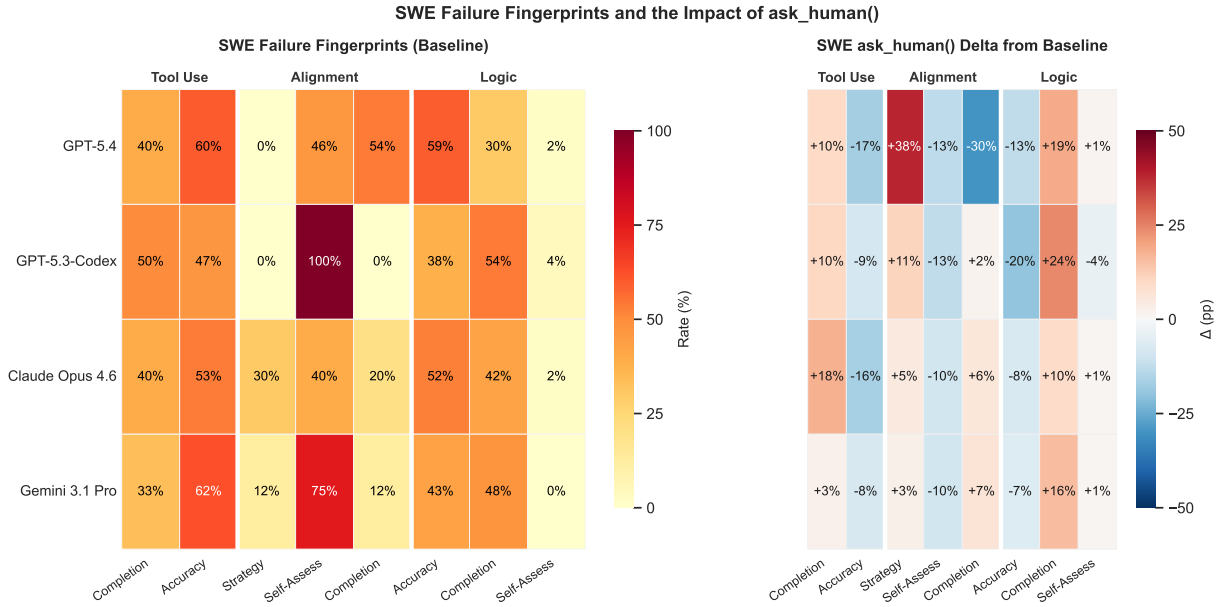


Figure 7: Failure modes in SWE. Compared to SQL, distributions congregate towards the same failure subtypes in all models

G RLVR Training Details

G.1 Model and Infrastructure

We fine-tune Qwen3 32B with LoRA using the SkyRL framework. Training is conducted separately for SQL and SWE domains, with 120 training tasks and 30 held-out evaluation tasks per domain. The agent operates within the same SWE-Agent framework and with the same tool access as the frontier model evaluations described in Sec. 4.1.

G.2 Reward Formulation

The total episode reward combines per-step and terminal components:

Per-step reward. Each `ask_human()` invocation receives:

$$r_{\text{step}}(q) = \begin{cases} +0.3 & \text{if } q \text{ targets a registered blocker} \\ -0.1 & \text{if } q \text{ is irrelevant or duplicates a resolved blocker} \end{cases} \quad (3)$$

The asymmetric magnitude (+0.3 vs. -0.1) encourages exploration of the question space while penalizing noise. This provides per-step signal analogous to precision.

Terminal reward. At episode completion:

$$r_{\text{terminal}} = \begin{cases} |B_{\text{discovered}}|/|B| & \text{if } |B_{\text{discovered}}| \geq 1 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where $B_{\text{discovered}} \subseteq B$ is the set of blockers for which the agent asked at least one relevant question. The gate condition prevents degenerate policies that avoid asking. Terminal coverage provides signal analogous to recall.

Total reward. $R = \sum_t r_{\text{step}}(q_t) + r_{\text{terminal}}$.

This decomposition preserves the Ask-F1 objective (per-step shapes precision, terminal shapes recall) while providing dense gradient signal that pure Ask-F1 as a terminal reward cannot.

H Complementary Metrics

In addition to Ask-F1 as the primary metric, we report complementary measures to provide a comprehensive view of agent behavior, including average questions asked per task (with the `ask_human` tool) and average tokens sent per task (for all three task conditions: baseline, full information, with tool).

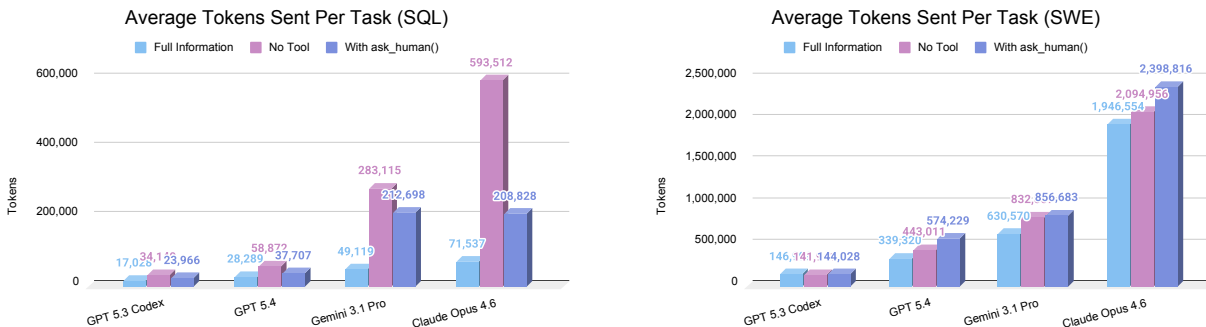


Figure 8: Average tokens used per task, per model. Model behaviors described above are also reflected here; GPT models execute quickly and overconfidently, using the least amount of tokens. Gemini sits more in the middle, while Claude explores, reflects, and explores some more, using up to five times as many tokens as other models.

Table 7: **Questions asked per task.** GPT models ask the lowest amount of questions in either domain. Gemini tends to over-ask on SQL. Claude sits more in the middle, but this doesn't necessarily translate to better integration of blocker resolutions into its workflow to solve tasks.

Model	SWE	SQL
GPT 5.3 Codex	1.5	1.0
GPT 5.4 Pro	1.8	1.7
Gemini 3.1 Pro	2.7	6.6
Claude Opus 4.6	4.7	4.5

I Full Leaderboard

Table 8: Pass@3 with `ask_human()` tool, combined across SWE and SQL.

Model	Combined				SWE				SQL			
	P@3	Prec	Rec	F1	P@3	Prec	Rec	F1	P@3	Prec	Rec	F1
Claude Opus 4.6	24	40	49	44	9	26	35	30	39	54	61	57
GLM-5.1	21	23	42	30	9	30	27	29	33	21	55	31
Gemini 3.1 Pro	20	39	48	43	5	48	36	41	35	36	59	44
GPT-5.4	9	53	25	34	1	52	27	36	17	54	23	32
Grok-4.20	8	19	28	23	7	17	35	23	9	22	22	22
Minimax-M2.5	7	41	12	19	1	36	2	3	14	41	22	28
GPT-5.3-codex	4	56	18	28	2	57	24	33	5	55	14	22