NL2FULLSTACK: EVALUATING DATA MODELING AND FULL-STACK APPLICATION GENERATION FROM NATURAL LANGUAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) coding agents can build complete database-backed applications from written briefs, but existing benchmarks grade user-visible workflows or the resemblance of a generated schema to a single reference, so backend business logic and data integrity are overlooked. We present NL2FullStack, a benchmark for data modeling and full-stack application generation from natural language. An agent receives a business brief and a binding engineering contract, then must ship a bootable application with a backend that implements the required workflows and a database that enforces the specified rules. Grading is deterministic and uses no LLM judge: the harness replays hidden, seeded workloads over the application's declared routes to test business logic, probes the database directly to verify integrity constraints, and checks that the interface displays backend rows. Across 89 tasks with five attempts each, Claude Opus 4.8 passes 62.9% of attempts, ahead of GPT-5.6 Sol (48.8%), Kimi K3 (44.5%), DeepSeek V4 Pro (29.2%), and Qwen3.8-Max (27.2%). The top four models pass the database tier on 87–96% of attempts, so nearly all the gap between models comes from the backend tier, where pass rates fall from 68% to 38%. For every model, the dominant failure is rejecting valid input under constraints absent from the contract. To mitigate contamination from public benchmarks, we release a nine-task public set with the grader, agent harness, and every graded attempt, and hold out eighty tasks as a private evaluation set.

1 INTRODUCTION

Today's enterprise software relies on relational databases to represent business operations and enforce their rules. Data modeling translates these rules into invariants that must hold whenever records are created, updated, or deleted: referential integrity, temporal validity, immutability after sign-off, and the distinction between stored and derived values. The data model expresses these requirements conceptually; the schema implements them through tables, keys, and constraints; and the backend exposes HTTP operations that must preserve them. Errors in this data layer are hard to detect and propagate silently. In one survey, roughly three quarters of production databases declared no foreign-key constraints (Kononow, 2018), and critical constraints are often implicit in a specification. An application may complete expected workflows while accepting writes that leave the database invalid, with corruption surfacing only later as production failures (Liu et al., 2024).

Large language model (LLM) coding agents increasingly implement application data layers, but current benchmarks rarely test whether their implementations preserve the invariants implied by a specification. Software-engineering benchmarks score code patches against hidden tests (Jimenez et al., 2024; Chowdhury et al., 2024) or evaluate tool use (Merrill et al., 2026; Yao et al., 2025). End-to-end application benchmarks evaluate observable behavior but generally do not test whether the underlying database enforces the required invariants (Zhong et al., 2026; Tran et al., 2026). Schema-generation benchmarks inspect the database more directly, but typically compare generated schemas with a single reference using string or set overlap (Wang et al., 2025a; Zheng et al., 2026), penalizing behaviorally correct alternatives while rewarding superficially similar schemas that fail to enforce the intended rules.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

NL2FullStack evaluates whether a running application faithfully implements the data requirements in its specification. Given a business brief and a binding engineering contract, an agent must deliver a bootable Node/Express application backed by PostgreSQL and accessible through a staff interface. Per-evaluation workload seeding limits reward hacking through hardcoded or memorized values.

This work makes three contributions. (i) To our knowledge, NL2FullStack is the first benchmark to grade the data model of a running, agent-built application rather than a schema in isolation or a user-visible workflow: a manifest-based protocol lets any correct design pass without a reference schema, and a three-tier harness independently tests database integrity, HTTP behavior, and interface wiring with no LLM judge. (ii) An 89-task benchmark spanning distinct business domains, every task validated against a control trio before any model runs, released with a nine-task public set, an eighty-task private set, a family-held-out evaluation protocol, the agent harness, and all graded attempts. (iii) An empirically grounded taxonomy of recurring data-modeling failures in frontier coding agents, enabled by tier-level diagnostics, whose central finding, that every model fails chiefly by rejecting valid input, reverses what the code-generation literature predicts. Together, these contributions establish a rigorous foundation for measuring and improving coding agents that build reliable, data-intensive full-stack applications.

2 Related Work

Appendix A compares related benchmarks in the coding and software application space.

Software agent benchmarks. SWE-bench and its Verified subset (Jimenez et al., 2024; Chowdhury et al., 2024) grade repository patches with hidden tests, Terminal-Bench (Merrill et al., 2026) scores terminal tasks with environment-specific suites, and τ-bench (Yao et al., 2025) compares the final database state of a tool-using agent with an annotated goal. All grade outcomes deterministically, but none examines the design or enforcement of the application's data model.

End-to-end application benchmarks. ViBench (Zhong et al., 2026), Vibe Code Bench (Tran et al., 2026), WebCoderBench (Liu et al., 2026), and VISTA (Guo et al., 2026) ask agents to build complete web applications and grade them through browser-driven workflows, LLM-based evaluators, or DOM and visual similarity; the best model reaches only 58.0% on Vibe Code Bench. These are the closest antecedents to our setting, but they grade observable pages and workflows, often with evaluators that add judge-dependent variance (Li et al., 2024; Gu et al., 2024), and they cannot tell a persisted value from a hardcoded or stale one, or a database-enforced invariant from an application-level check.

Database and schema benchmarks. *Schema-as-input* benchmarks supply the database structure and score a task over it: text-to-SQL (Yu et al., 2018; Li et al., 2023; Lei et al., 2025), constraint-derived question answering (Oh et al., 2024), schema matching (Parciak et al., 2024), and data-product construction (Chowdhury et al., 2025). Because the schema is given, they do not test schema design. *Schema-as-output* benchmarks score a generated schema against a reference: SchemaAgent (Wang et al., 2025a) and LLM database-design tools (Divljan & Brdjanin, 2025) compare tables, attributes, keys, and types; DBLifeBench (Zheng et al., 2026) scores entity names, types, and foreign keys; and DNBENCH (Koh et al., 2026) checks normalized decompositions from 1NF to BCNF. These treat the schema as a standalone artifact rather than as the persistence layer of a running application, and reference similarity is neither necessary nor sufficient for correctness: a different schema may enforce every required invariant while a near-identical one omits the constraints.

3 The NL2FullStack Benchmark

3.1 Problem Formulation

Task. A task is a tuple $T = (B, C, G, W)$. The business brief B is an operator-voice narrative of how the business runs and the agent's only source of domain rules. The engineering contract C is the binding specification the agent builds to: platform, logical operations with argument and return names and wire formats, self-declaration format, authentication, and the public site. The gold implementation G is a hidden reference build, and the workload generator W produces seeded, randomized gold workloads. The agent sees only (B, C) in a sandbox with PostgreSQL and a

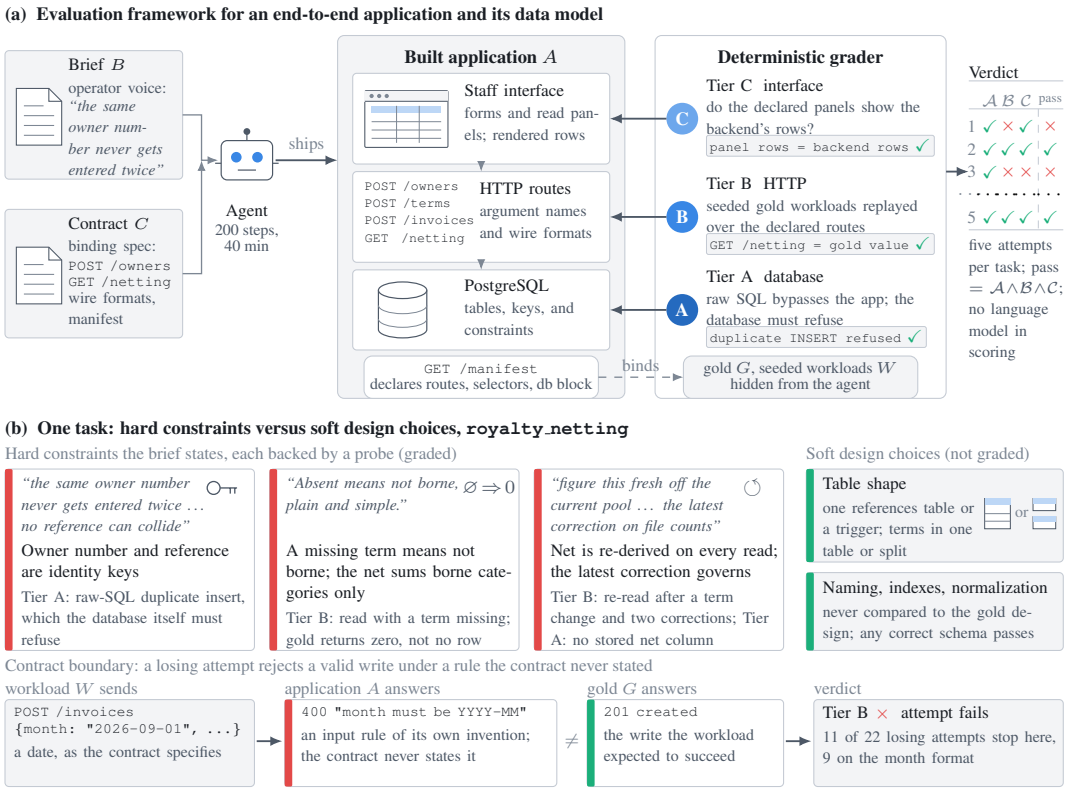

Figure 1: **Evaluating an agent-built application and its data model.** (a) The agent receives a brief B and a contract C and ships an application A; the grader binds to A's manifest and verifies it from outside at three surfaces, and an attempt passes only if all three tiers pass (verdict checklist schematic). (b) `royalty_netting`: three sentences of the brief become three probed hard constraints, while table shape and naming stay soft design choices the grader never inspects, and the trace shows an attempt failing Tier B on a month format the contract never stated.

runtime and must produce an application A that boots via `run.sh` (Figure 1a); G, W, and every probe stay with the grader, and because W is re-seeded per evaluation, expected outputs cannot be memorized. Appendix B lists all notation and terminology used in this work.

Graded properties. An *invariant* is a rule about the data that must hold whichever code path wrote it: an owner number is entered once, a signed figure never changes, a net is derived rather than stored. A *hard constraint* is an invariant the brief states or directly implies, and every one is backed by a deterministic probe (the red cards in Figure 1b, each pairing a brief sentence with its probe). They come in two kinds, and every task carries both: workflow rules (which correction governs, what an absent record means, how a figure is derived, what must be refused) and integrity rules (identity keys, references, immutability). A *soft design choice* is anything the contract leaves open, such as table shape, naming, normalization, and indexing (the green cards); it is never graded directly, and any choice that behaves correctly passes. Two soft-looking choices are graded anyway, because both are behavioral hazards a deterministic probe can catch: storing a derivable fact redundantly, and entity–attribute–value over-sharing (§3.2). The benchmark therefore measures whether a model (a) implements every hard constraint of both kinds and (b) refrains from inventing hard constraints the contract does not state; the trace at the bottom of Figure 1b shows an attempt failing on (b).

Self-declaration. The contract requires A to publish a manifest $M(A) = (R, \Sigma, D)$ at `GET /manifest`. R lists the HTTP routes the application exposes and the argument names each accepts; Σ names the interface panels the grader may inspect; and the `db` block D points to the tables and columns that hold protected values (frozen figures and unique keys), so Tier A knows where

to probe. The grader discovers everything it probes from $M(A)$, so nothing about the gold design enters grading.

Pass criterion. Grading is a deterministic predicate over three independent surfaces,

$$\text{pass}(A) \;=\; \underbrace{\mathcal{A}(A)}_{\text{database}} \wedge \underbrace{\mathcal{B}(A)}_{\text{HTTP}} \wedge \underbrace{\mathcal{C}(A)}_{\text{interface}}, \tag{1}$$

with the tiers defined in §3.2.

Metrics. Every task is attempted five times, and each metric answers one question. The headline *pass rate*, the fraction of a task's attempts that pass averaged over tasks, measures raw difficulty; averaging over tasks weights every task equally and gives a bootstrap uncertainty estimate over tasks (10,000 resamples, 95% percentile interval). *pass@5*, the fraction of tasks solved at least once (Chen et al., 2021), measures the ceiling with retries. The *per-tier pass rate* says which surface moved. A task's *pass count* $c \in \{0, \ldots, 5\}$ is its number of passing attempts, and its *band*, never ($c = 0$), sometimes ($1 \leq c \leq 4$), or always ($c = 5$) solved, separates tasks that discriminate between models from those that do not. Two *direction diagnostics*, over-permissive probe failures (a write accepted where gold rejects) and over-strict ones (a write rejected where gold accepts), counted once per workflow at its first failure so one early rejection does not inflate the count, expose the mechanism of failure; their ratio is the benchmark's most diagnostic single statistic (§4.3). Appendix C tabulates every metric with its confidence interval.

3.2 Grading Harness

The harness composes standard testing ideas, behavioral checks in the tradition of metamorphic, property-based, and differential testing (Chen et al., 2018; Claessen & Hughes, 2000; McKeeman, 1998), execution grading (Ren et al., 2020), per-evaluation workload randomization (Qian et al., 2024), and taint-style value provenance (Schwartz et al., 2010), and applies them jointly to the data layer of a running application.

Tier A: database enforcement. An integrity rule enforced only in application code is no integrity rule, since any concurrent writer, migration script, or bug that touches the database directly can violate it. Tier A therefore connects to PostgreSQL and attempts violations with raw data-manipulation-language (DML) statements, inserting a duplicate under a declared identity key, mutating a signed record, orphaning a child row, and passes only if the database itself refuses. The structure scan seeds distinctive values through the application's own write path and searches the physical schema for unexplained copies (a value-provenance dye), which finds redundant storage of derivable facts without pattern-matching table or column names.

Tier B: business logic over HTTP. The grader runs the hidden gold implementation on the seeded workload to obtain expected outputs, replays the same workload against the submission over its declared routes, and passes it when the results match: values compared exactly under the canonicalization the contract discloses (§5), money at cent precision and rates at full precision. Gold variants exist only at authoring time, to certify that a task admits more than one design; the grader never matches a submission against them. Replay covers happy-path, rejection, and anomaly workloads, so the tier catches wrong derived values, wrong accept-or-reject decisions, stale caches, and self-invented input validation.

Tier C: interface wiring. The interface tier asks one question: is this panel connected to the backend? Each backend row must claim a distinct rendered row through per-row anchor evidence, distinctive values from the backend's actual response, tolerant of rendering choices such as thousands separators, currency symbols, date formats, nulls shown as dashes, labeled enums, or extra columns; weak evidence (argument echoes, values shared by every row, bare single digits, hidden-node text) does not count. Row-count and fabrication checks and a check that rejected writes surface a refusal complete the tier. Exact value verification is Tier B's job, where no rendering layer intervenes; this tier catches hardcoded panels, unwired reads, and silent success on a rejected write.

Anti-gaming spine. The answer key never enters the build workspace; the sandbox has no grader access; Tier A cannot be satisfied by application code; the structure scan blocks the giant-denormalized-table strategy that would otherwise satisfy Tier B; and Tier C's evidence tiers resist echo-stuffing.

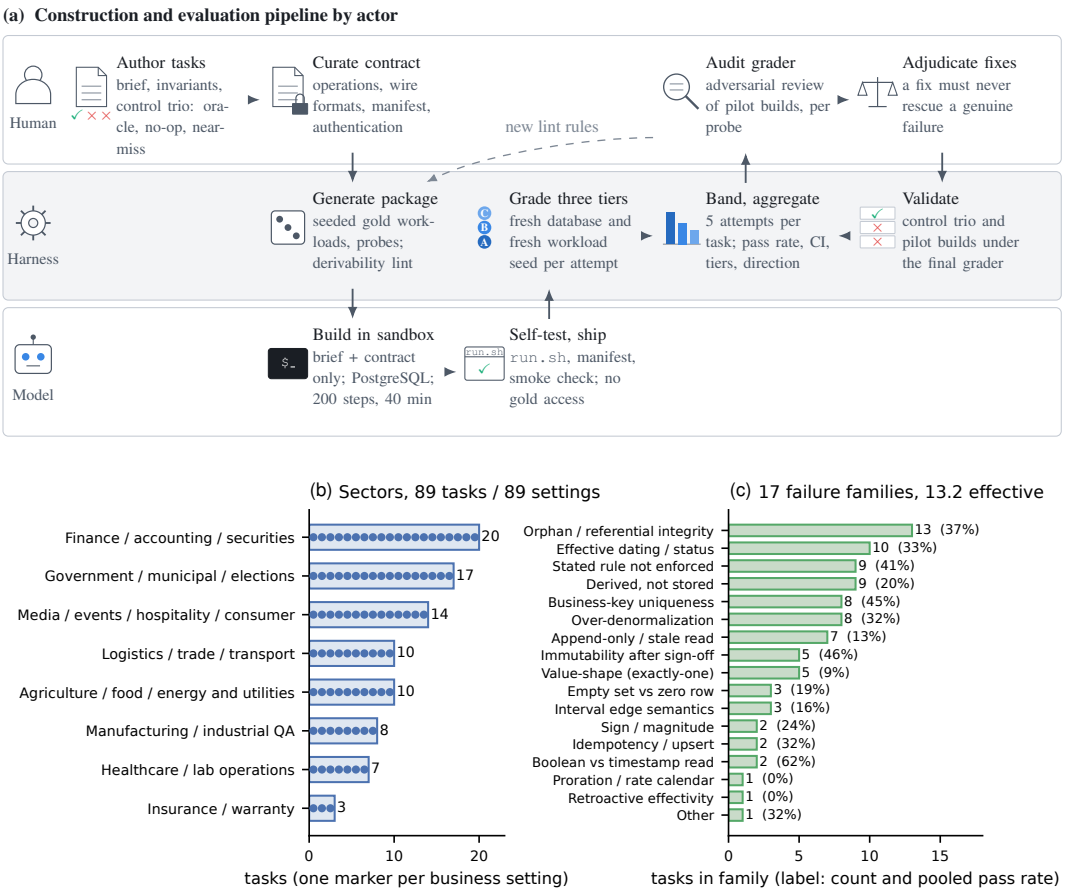

Figure 2: **Construction pipeline and corpus.** (a) Pipeline by actor: humans author tasks and audit the grader, the harness generates packages and grades, and the model builds from the brief and contract alone; audit findings feed lint rules back into generation (§5). (b) Number of tasks per sector, one marker per task (every task is a different business setting). (c) The 17 failure families: each task is written to catch one specific data-modeling mistake, grouped into families. Bar: number of tasks targeting the family. Percentage: pass rate on that family's tasks, all five models pooled; lower means harder.

3.3 Benchmark Construction

Figure 2 shows the pipeline by actor. The benchmark contains 89 tasks (a ninetieth was retired in quality assurance), each authored with a control trio, a hidden oracle implementation that must pass and no-op and near-miss implementations that must fail, so every grader is validated against known-good and known-bad designs before any model runs.

Human roles. Humans appear in three roles: authoring and curating tasks (brief, contract, control trio, and probes), adversarial QA of the grader (the pilot audits of §5), and adjudicating fixes.

Corpus. Each task package ships the brief and contract, the deterministic grader, the seeded gold workload generator, and per-attempt grade records and build trajectories for the five reference models. The 89 tasks span 89 distinct business settings across eight sectors (Figure 2b; Appendix D), so schema vocabulary cannot be pattern-matched from one task to the next. Skill breadth is measured separately: each task elicits one of 17 failure families (Figure 2c; Appendix E), with an effective diversity (perplexity of the family distribution) of 13.2 families. Roughly two thirds are workflow rules (effective dating, derived-versus-stored values, corrections and stale reads, empty-set-versus-zero-row contracts, interval edges, sign conventions, proration) and one third database-hygiene rules (referential integrity, business-key uniqueness, immutability after sign-off, over-denormalization).

Pooled difficulty ranges from 62% of attempts passing on the boolean-versus-timestamp read contract to 9% on the exactly-one value-shape contract.

Example task. The `royalty_netting` brief describes netting post-production costs out of royalty payments (Appendix F gives its operative paragraphs and full probe set). Figure 1b traces it through the harness: three brief sentences become three hard constraints, each with a Tier A or Tier B probe, while table shape is left to the agent. Its workflow rules have sharp edges: "the latest correction on file is the one that counts" makes the pool total a function of the correction history rather than the invoice table, and "absent means not borne" is an empty-set-versus-zero contract that an inner join silently violates. Yet 11 of the task's 22 losing attempts never reached those rules; they rejected the happy path's first write because the workload sent the month as a date, as the contract specifies, and the application demanded `YYYY-MM`.

Public/private partition. Nine tasks form the public set and eighty are held out as a family-stratified private set, which limits contamination of the evaluation set; all numbers in this paper use all 89. The nine public tasks and the evaluation code are included in the supplementary material, and Appendix G lists the released artifacts and the family-held-out splits for adaptation experiments.

4 Experiments

4.1 Evaluation setup

Models. We evaluate five models spanning the frontier and the open-weight tier: Claude Opus 4.8, Kimi K3 (reasoning effort high), GPT-5.6 Sol (reasoning effort max, Responses API), DeepSeek V4 Pro (reasoning effort high), and Qwen3.8-Max (thinking budget 4,000 tokens). Each model's reasoning setting is the highest its API exposes through the harness; full configuration is in Appendix H.

Harness and protocol. All models are driven by the OpenHands coding agent (Wang et al., 2025b) (OpenHands CLI 0.62.0, native function calling) with a fixed budget of 200 steps and a 4-hour wall-clock cap per attempt (no attempt reached the cap). Every task receives five fresh builds per model, each in an ephemeral sandbox, graded on a fresh database with a fresh workload seed.

4.2 Results

Figure 3 reports the headline comparison; Appendix C tabulates it. Claude Opus 4.8, run at maximum reasoning effort, leads with a 62.9% pass rate (95% CI $[55.1, 70.5]$), followed by GPT-5.6 Sol at 48.8% $[40.7, 57.1]$, Kimi K3 at 44.5% $[37.3, 51.7]$, DeepSeek V4 Pro at 29.2% $[23.4, 35.3]$, and Qwen3.8-Max at 27.2% $[21.8, 32.8]$. Only Sol's interval overlaps the leader's; Sol and Kimi overlap almost entirely, and DeepSeek and Qwen are indistinguishable. Retries change the ordering: Kimi solves 66 tasks at least once against Sol's 62, so Sol's higher pass rate reflects reliability on the tasks it can solve rather than a larger solvable set (pass$\hat{}$5: Sol 22%, Kimi 15%).

Per-tier decomposition. The per-tier decomposition (Figure 3b) places nearly all of the separation in Tier B, the business-logic tier. Tier A passes on 87–96% of attempts for the top four models and Tier C on 84–93% for three of them, while Tier B ranges from 68% (Opus) down to 38% (Qwen). Two models have a second bottleneck: DeepSeek passes Tier C on only 65% of attempts, and Qwen produced no runnable application in 31%.

Bands and the task set. For every model, at least 42 of 89 tasks are in-band ($1 \leq c \leq 4$), the region that carries discriminative signal; five tasks are in-band for all five models. Thirty-eight tasks are solved at least once by every model and eight by none (Figure 3d; the eight are listed in Appendix I). The unsolved eight span six failure families.

4.3 Failure signatures

Over-strictness dominates. Each losing attempt is assigned one category, its first failing check in tier order (no runnable application, then Tiers A, B, C), by a fixed rule over the grade record (Figure 4a). Rejecting a contract-valid write is the most common first failure for four of the five models, about half of their losing attempts, and second only to producing no runnable application

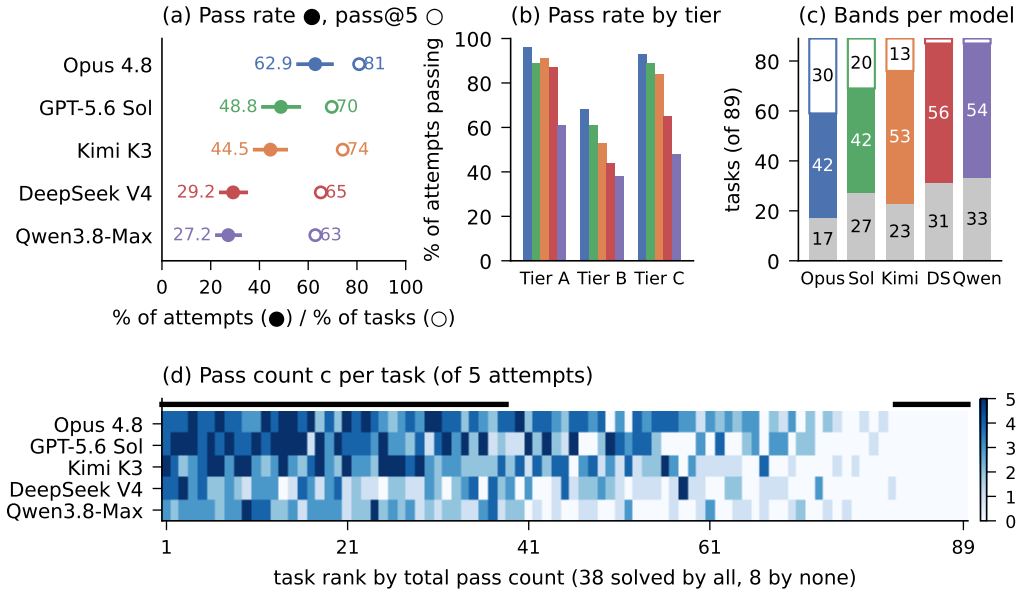

Figure 3: **Evaluation results on frontier models.** 89 tasks, five attempts each. (a) Pass rate with a 95% bootstrap confidence interval over tasks (filled) and pass@5, the fraction of tasks solved at least once (open). (b) Fraction of graded attempts passing each tier: the top four models pass Tier A on 87–96% of attempts and the top three pass Tier C on 84–93%, while Tier B separates them, from 68% down to 38%. (c) Band distribution per model: never solved (grey), in-band (color), always solved (white). (d) Pass count c per task and model, tasks sorted by total pass count; black bars mark the 38 tasks solved at least once by every model and the 8 solved by none.

for Qwen. Counted per workflow, over-strict Tier-B failures outnumber over-permissive ones by 20 to one (DeepSeek) up to 160 to one (Qwen) (Figure 4b). Wrong read results are the second failure (20–29% of losing attempts; Qwen 11%), and database-tier failures are a minority everywhere (10–21%). Prior work on generated code reports the opposite failure, missing checks rather than extra ones (Li et al., 2025; Pearce et al., 2022); here the direction is reversed for every model.

Over-strictness in code. A forensic audit tied every failing Opus 4.8 attempt to a root cause in the built application and trajectory, each finding adversarially verified; its records ship with the dataset. The recurring patterns are whole-number gates on money, two-decimal regular expressions, `YYYY-MM` month formats where the contract sends a date, strict ISO 8601 parsing, and `typeof`-string checks on contract-typed numbers. The largest class of rejection message (96 of 484) is rule-shaped: the application turns a business rule that the brief states but the contract never operationalizes into a 409, and the gold workload then performs the action the sentence seemed to forbid. Appendix J gives the full decomposition.

Per-build variance. Most in-band tasks sit at a pass count of one or two for every model (Figure 4c): the same model builds the correct data model in some attempts and ships a rejected write or a malformed query in the others. The $n{=}5$ protocol keeps this per-build variance on purpose: a backend benchmark should measure whether a correct system is shipped reliably.

Coverage caveat. Where over-strictness fires before the designed trap, the loss is a genuine backend failure but the task's data-model competency goes unmeasured for that attempt, so headline scores should be read next to Figure 4a. Canonicalizing inputs before grading would recover pure data-model signal, but we keep input discipline graded: a backend that rejects contract-valid input is broken.

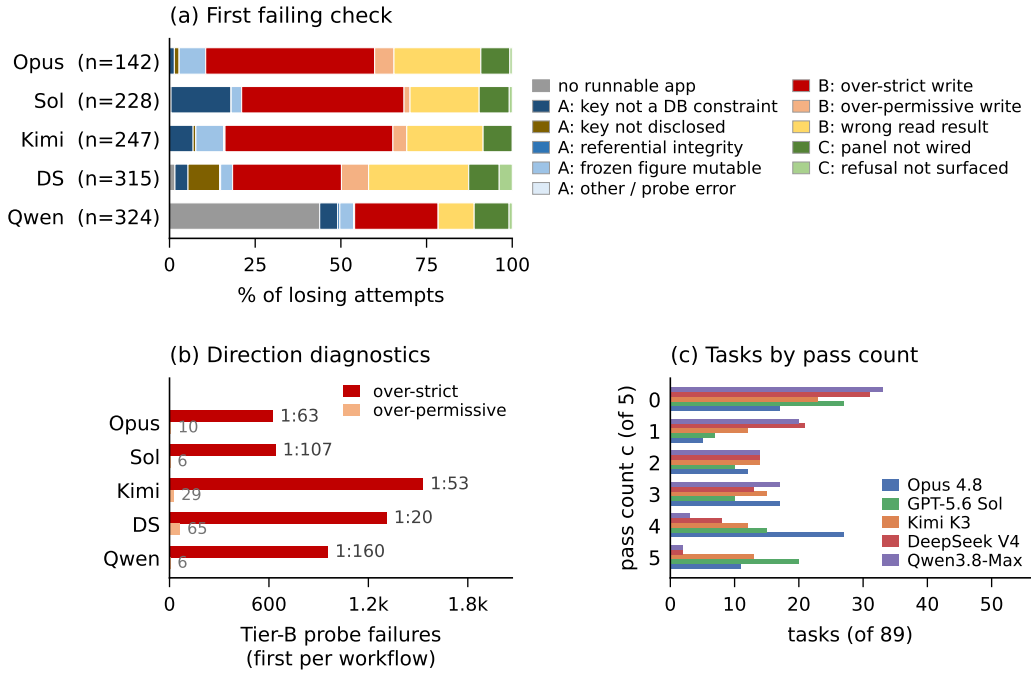

Figure 4: **First-failure decomposition of losing attempts.** (a) The first failing check of every losing attempt, assigned deterministically from the grade record in tier order (no runnable application; then Tier A, B, C), with each model's number of losing attempts in parentheses. (b) Tier-B probe failures by direction, counted once per workflow at its first failure; the label is the over-permissive : over-strict ratio. (c) Number of tasks at each pass count c.

Table 1: Cross-harness comparison on a 20-task stratified subset, five attempts per model per harness, same frozen grader. Pass rates carry 95% bootstrap intervals; Δ is the mean per-task difference (OpenCode − OpenHands) with its interval, and $\uparrow/\downarrow/=$ counts tasks where OpenCode is higher, lower, or tied.

	Pass rate (95% CI)		Paired Δ	Tasks	pass@5		No runnable app	
Model	**OpenHands**	**OpenCode**	**(OC−OH)**	$\uparrow/\downarrow/=$	**OH**	**OC**	**OH**	**OC**
Claude Opus 4.8	61 [45, 75]	65 [49, 80]	+4 [−8, +17]	7/5/8	85	90	0%	0%
Kimi K3	49 [33, 66]	52 [37, 66]	+3 [−9, +14]	8/7/5	80	80	0%	4%
GPT-5.6 Sol	40 [23, 58]	47 [29, 66]	+7 [−2, +18]	6/4/10	65	70	1%	0%
DeepSeek V4 Pro	35 [21, 50]	36 [24, 49]	+1 [−11, +14]	6/7/7	60	80	0%	0%
Qwen3.8-Max	32 [20, 44]	50 [34, 66]	**+18** [+5, +32]	12/4/4	65	80	28%	0%

4.4 Cross-harness comparison

Because the coding-agent harness is part of the system under test (§4.1), we rebuilt all five models under a second agent, the OpenCode CLI, on a stratified random 20-task subset with the same budgets, endpoints, and frozen grader (Table 1; protocol in Appendix K). The harness does not change the conclusions: the model ordering is identical under both agents, and four of the five paired differences sit inside their 95% intervals (Opus +4, Kimi +3, Sol +7, DeepSeek +1 points). The exception is Qwen3.8-Max at +18 points, and the mechanism is reliability rather than reasoning: under OpenHands, Qwen produced no runnable application on 28% of these attempts, under OpenCode on none. The takeaway is that a harness change is a floor-raising operation for weak models and close to a no-op for strong ones: the business-logic separation reported above is a property of

the models, while lower-half leaderboard scores are a model-times-harness product until harness affordances are equalized.

5 Grader Validity: Artifact Classes and Controls

A hidden test cannot be argued with, yet the same machinery can manufacture difficulty unrelated to the intended skill, so we treat the grader as an object of study. Before the evaluation we ran a pilot in which every failing agent build was adversarially reviewed against the probe it failed. Three rounds each isolated an artifact class, produced a fix, and verified it with controls (Appendix L). The classes were: *rendering-layer value re-checking* (exact-value comparison through the rendering layer, so commas, null dashes, and labeled enums read as "not wired"; the fix is the wiring-only Tier C of §3.2); *undisclosed grader dialect* (gold values serialized in forms the contract never stated, such as scientific notation or undisclosed enum tokens; the fix is value-identical normalization plus disclosure in the contract); *comparison intolerance* (correct-but-unrounded money folds failing 10^{-4} equality; the fix is a strict-first compare with a cent-rounding retry); and *verdict bookkeeping* (a stored pass verdict that can disagree with the three tier booleans; the fix makes the tier booleans the single source of truth).

Verification discipline. Every fix was checked two ways. It must keep each task's control trio intact (oracle passes; no-op and near-miss fail) and must not *over-rescue*: clearing an artifact must leave the genuine failures standing. The decisive over-rescue control was an authentication-shape task: the login fix cleared every spurious 401 in the pilot builds and rescued none of them, leaving the genuine domain rule standing. All attempts reported in §4 were graded once, under the frozen grader.

Contract-derivability lint. All dialect artifacts are one disease: the generator or grader speaking a dialect the contract never disclosed. The pre-emptive gate is one build-time check: every value the grader will send or compare against must be derivable from the contract text. The lint is now in the construction pipeline (Figure 2) and would have caught the whole class before any model ran.

6 Discussion

Difficulty originates from rules, not domains. What discriminates in-band is business logic (temporal and effective-dating rules, corrections and stale reads, empty-set-versus-zero-row contracts, sign conventions) and database hygiene (scoped uniqueness, frozen-provenance capture), plus per-build engineering variance. The business-logic tier, where the specification must be honored exactly as written, is where models separate.

Stated versus implied specification. An implied regime measures inference under under-specification; a stated regime measures faithful implementation of an explicit contract, an axis that looks saturated until every model is seen violating the contract from the strict side (§4.3). NL2FullStack is contract-explicit by design; the implied axis enters through the planned product-isomorph families (Appendix M).

Limitations and future work. Bands and failure signatures are relative to specific models under a specific harness, over-strictness can pre-empt a designed trap, and the checks are property tests rather than proofs; extending the harness ladder to more agents and adapting on the public set with evaluation on held-out failure and product families are the natural next steps (Appendix M).

7 Conclusion

NL2FullStack's thesis is that data-model correctness is behaviorally decidable, deterministically and with no language model on the accept path. The three-tier harness grades business logic and data integrity together. The five-model study separates frontier and open-weight models with confidence intervals, attributes the separation to the backend tier, and shows that every model's dominant failure is to reject input the contract allows. We publish the artifact taxonomy, control protocol, and derivability lint, and release nine public tasks with the grader, the harness, and every graded attempt, holding out eighty as a private set.

Ethics Statement

NL2FullStack contains no personal data. Every business setting, brief, contract, and gold implementation was authored by the authors, and every workload value is generated from a seed at evaluation time, so the released tasks, grade records, and build trajectories contain no real customer, employee, or transaction records. No human-subjects research was conducted: all tasks were authored, audited, and adjudicated by the authors, so no crowd workers or paid annotators were involved and no consent or compensation questions arise. The released artifacts carry the open license stated in the supplementary data card. The evaluated models were accessed through their public APIs under the providers' terms of service, and no model was fine-tuned on benchmark material. To limit contamination, eighty of the 89 tasks are withheld from public release and served only through a held-out evaluation protocol. The benchmark measures whether generated applications enforce stated data-integrity rules; we see no direct dual-use risk beyond that of coding benchmarks generally, and a foreseeable benefit in surfacing silent data-corruption failures before agent-built software reaches production. The use of language models in preparing this paper is disclosed in the statement on the use of large language models below.

Use of Large Language Models

Large language models are the subjects of this benchmark: the reference runs evaluate five models as application builders. Language-model agents were also used in the forensic audit of the Opus 4.8 run to propose code-level root causes; every proposed root cause was checked against the built application and the builder's trajectory by two authors before inclusion. Language models assisted with editing prose and with figure and table formatting; all text was reviewed by the authors, and every reference was verified by hand against the publisher or arXiv record. No language model participates in grading, and no language model produced any result, number, or interpretation reported here. All benchmark design, task authoring, audits, and analyses are the authors' responsibility.

Reproducibility Statement

All quantitative claims trace to per-attempt grade records (per tier, per probe), the forensic findings file from the Opus 4.8 audit, and the pilot-audit records; one script regenerates every figure and table in this paper from those records. The nine public tasks and the evaluation code are included in the supplementary material, and the public set with its data card is hosted at an anonymized repository (link to be added at submission). The evaluation configuration of every run is in Appendix H, the headline metrics with confidence intervals in Appendix C, per-task results in Appendix I, the public/private sets and the held-out scoring protocol in Appendix G, and the grader-validation controls in Appendix L. The public set packages the nine tasks in a standard agent-evaluation task format with a one-command runner for the exact configurations of Appendix H, a re-grading script that reproduces any shipped verdict from the shipped application, and every trajectory on those tasks with its built application. The eighty private tasks are evaluated by the authors under the same runner.

References

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 preprint arXiv:2107.03374*, 2021.

Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys*, 51(1), 2018.

Faisal Chowdhury, Sola Shirai, Sarthak Dash, Nandana Mihindukulasooriya, and Horst Samulowitz. Bridging business intent and data: A benchmark for automatic relational data product generation. *arXiv preprint arXiv:2512.15798*, 2025.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified. OpenAI blog post, 2024.

Koen Claessen and John Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of ICFP*, 2000.

Petar Divljan and Drazen Brdjanin. Towards an LLM-based tool for automated database design. In *ER 2025 Companion Proceedings, CEUR-WS Vol. 4099*, 2025.

Iker García-Ferrero, Begoña Altuna, Javier Álvez, Itziar Gonzalez-Dios, and German Rigau. This is not a dataset: A large negation benchmark to challenge large language models. In *Proceedings of EMNLP*, 2023.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11), 2020.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on LLM-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

JunJia Guo, Yuhang Yao, Jiawei Zhou, and Jingdi Chen. VISTA: An end-to-end benchmark for visual spec-to-web-app coding agents. *arXiv preprint arXiv:2605.26144*, 2026.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Dong-Jae Koh, Huisu Kim, SeongHwan Yoon, Lasse M. Jantsch, Chun-Hee Lee, Seonghyeon Lee, and Young-Kyoon Suh. Can LLMs normalize databases? a benchmark and multi-agent framework for schema normalization. *arXiv preprint arXiv:2609.11141*, 2026.

Pawel Kononow. Don't most databases have foreign key constraints? Dataedo Blog, 2018.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: The flip side of AI ingenuity. DeepMind blog post, 2020.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, 2025.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. LLMs-as-judges: A comprehensive survey on LLM-based evaluation methods. *arXiv preprint arXiv:2412.05579*, 2024.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023.

Zike Li, Mingwei Liu, Anji Li, Kaifeng He, Yanlin Wang, Xin Peng, and Zibin Zheng. A preliminary study on the robustness of code generation by large language models. *arXiv preprint arXiv:2503.20197*, 2025.

Bihui Liu, Saikat Mondal, and Tse-Hsun Chen. An empirical study on the characteristics of database access bugs in Java applications. *ACM Transactions on Software Engineering and Methodology*, 2024.

Chenxu Liu, Yingjie Fu, Wei Yang, Ying Zhang, and Tao Xie. WebCoderBench: Benchmarking web application generation with comprehensive and interpretable evaluation metrics. *arXiv preprint arXiv:2601.02430*, 2026.

William M. McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1), 1998.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.

Jio Oh, Soyeon Kim, Junseok Seo, Jindong Wang, Ruochen Xu, Xing Xie, and Steven Euijong Whang. ERBench: An entity-relationship based automatically verifiable hallucination benchmark for large language models. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024.

Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M. Peeters, and Stijn Vansummeren. Schema matching with large language models: An experimental study. *Proceedings of the 2nd International Workshop on Tabular Data Analysis (TaDA), co-located with VLDB*, 2024.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of GitHub copilot's code contributions. In *IEEE Symposium on Security and Privacy*, 2022.

Kun Qian, Shunji Wan, Claudia Tang, Youzhi Wang, Xuanming Zhang, Maximillian Chen, and Zhou Yu. VarBench: Robust language model benchmarking through dynamic variable perturbation. *arXiv preprint arXiv:2406.17681*, 2024.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. CodeBLEU: A method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Scale AI (SEAL Labs). SWE-bench pro leaderboard (public and private datasets). https://scale.com/leaderboard, 2025.

Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *IEEE Symposium on Security and Privacy*, 2010.

Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, Shauna Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models. In *International Conference on Learning Representations*, 2024.

Hung Tran, Langston Nashold, Rayan Krishnan, Antoine Bigeard, and Alex Gu. Vibe Code Bench: Evaluating AI models on end-to-end web application development. *arXiv preprint arXiv:2603.04601*, 2026.

Qin Wang, Youhuan Li, Yansong Feng, Si Chen, Ziming Li, Pan Zhang, Zihui Si, Yixuan Chen, Zhichao Shi, Zebin Huang, Guo Chen, and Wenqiang Jin. Text2Schema: Filling the gap in designing database table structures based on natural language. *arXiv preprint arXiv:2503.23886*, 2025a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of EMNLP*, 2018.

Sicheng Zheng, Deyan Shi, Yi Li, Xudong Yi, Lei Wang, and Gerard de Melo. Evaluating LLMs in database scenarios: A lifecycle benchmark for assessing their potential in core database tasks (DBLifeBench). *arXiv preprint arXiv:2608.03794*, 2026.

Peter Zhong, Pashootan Vaezipoor, Fuyang Cui, Vaibhav Kumar, James Austin, Azin Asgarian, Toby Ho, Paul Inder, Imen Kedir, Zhen Li, Nicholas Ondo, Asna Shafiq, Ibrahim Sheikh, Edouard Sioufi, Setareh Soltanieh, Ben Wilde, Jacky Zhao, Ryan Carelli, Heather Miller, and Michele Catasta. ViBench: A benchmark on vibe coding. In *Proceedings of the ACM Conference on AI and Agentic Systems (CAIS)*, 2026.

A Related-Benchmark Comparison

Table 2 compares related benchmarks in the coding and software application space.

Table 2: Benchmarks along the dimensions relevant to assessing an agent-built data layer. *Graded surfaces*: whether scoring directly examines the database schema, stored state, or constraints (*DB*); HTTP request–response behavior (*HTTP*); and user-visible behavior (*UI*). *Deterministic*: a predefined, non-LLM oracle – tests, execution results, or state comparison – so the same submission always receives the same verdict. *Ref.-free*: evaluation that does not compare against a single reference implementation or schema. ✓ = direct coverage, ∼ = partial or indirect, × = none; symbols describe what the score depends on, not what the task may access.

	Graded surfaces			**Evaluation oracle**	
Benchmark	**DB**	**HTTP**	**UI**	**Deterministic**	**Ref.-free**
Software agents					
SWE-bench (Jimenez et al., 2024)	×	×	×	✓	✓
SWE-bench Verified (Chowdhury et al., 2024)	×	×	×	✓	✓
Terminal-Bench (Merrill et al., 2026)	×	×	×	✓	✓
τ-bench (Yao et al., 2025)	✓	×	×	✓	✓
End-to-end applications					
ViBench (Zhong et al., 2026)	×	∼	✓	×	✓
Vibe Code Bench (Tran et al., 2026)	×	∼	✓	×	✓
WebCoderBench (Liu et al., 2026)	×	×	✓	∼	✓
VISTA (Guo et al., 2026)	×	×	✓	✓	×
Schema-as-input					
Spider (Yu et al., 2018)	∼	×	×	✓	×
BIRD (Li et al., 2023)	∼	×	×	✓	×
Spider 2.0 (Lei et al., 2025)	∼	×	×	✓	×
ERBench (Oh et al., 2024)	∼	×	×	✓	×
Schema matching (Parciak et al., 2024)	×	×	×	✓	×
DP-Bench (Chowdhury et al., 2025)	∼	×	×	✓	×
Schema-as-output					
SchemaAgent (Wang et al., 2025a)	∼	×	×	✓	×
LLM database-design tools (Divljan & Brdjanin, 2025)	∼	×	×	✓	×
DBLifeBench (design) (Zheng et al., 2026)	∼	×	×	✓	×
DNBENCH (Koh et al., 2026)	∼	×	×	✓	∼
NL2FullStack (ours)	✓	✓	✓	✓	✓

B TERMINOLOGY AND NOTATION

Table 3 collects the terms and symbols used throughout the paper.

Table 3: Terminology and notation used throughout the paper.

Symbol	Term	Meaning in this paper
T	task	the tuple $T = (B, C, G, W)$ defining one benchmark task
B	brief	the operator-voice narrative of the business; the model's only source of domain rules
C	contract	the binding engineering specification: platform, operations with argument and return names, manifest format, authentication, public site
G	gold implementation	a hidden reference build, never shown to the agent; used to derive expected workload outcomes, never compared against the submission's schema
W	workload generator	seeded and randomized per evaluation; produces the gold workloads Tier B replays, so expected outputs cannot be memorized
A	application	one submitted build: schema, backend, and interface, bootable via `run.sh`
$M(A)$	manifest	the application's self-declaration at `GET /manifest`, equal to (R, Σ, D); the grader discovers everything it probes from it
R	HTTP routes	declared operations: method, path, argument names, return names
Σ	interface selectors	the declared handles by which Tier C locates panels and fields
D	`db` block	the part of the manifest locating frozen figures and unique keys in the application's own physical schema
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	tiers	the database, HTTP, and interface surfaces; an attempt passes when $\mathrm{pass}(A) = \mathcal{A}(A) \wedge \mathcal{B}(A) \wedge \mathcal{C}(A)$
c of n	band	a task's classification for a model from c passes of n attempts: never solved (c=0), in-band (0<c<n), always solved (c=n)
	invariant	a rule about the data that must hold at all times, stated or implied by the brief (identity keys, references, effectivity, immutability after sign-off)
	hard constraint	an invariant the grader can fail deterministically; each is backed by a Tier-A or Tier-B probe
	soft design choice	any decision the contract leaves open (table shapes, naming, normalization strategy, indexing); never graded directly
	workload	a seeded sequence of writes and reads with gold results, generated by W and replayed by Tier B
	probe	one grader action with a deterministic expected outcome (a raw SQL write that must be refused, a request whose result must match)
	attempt	one build of one task by one model in a fresh sandbox; graded on a fresh database

C HEADLINE METRICS

Table 4 reports every headline metric for each model over the 89 tasks, with bootstrap confidence intervals and the per-tier decomposition. pass@5 follows Chen et al. (2021); with five attempts per task it is simply the fraction of tasks solved at least once.

Table 4: Headline metrics per model on 89 tasks, five attempts each. Confidence intervals are 95% percentile intervals from 10,000 bootstrap resamples over tasks. Direction diagnostics are Tier-B probe failures, first per workflow and in total.

Metric	Opus 4.8	GPT-5.6 Sol	Kimi K3	DeepSeek V4	Qwen3.8-Max
pass rate (mean over tasks of passes / 5)	62.9%	48.8%	44.5%	29.2%	27.2%
95% CI	[55.1, 70.5]	[40.7, 57.1]	[37.3, 51.7]	[23.4, 35.3]	[21.8, 32.8]
pass@5 (tasks solved at least once)	80.9% (72/89)	69.7% (62/89)	74.2% (66/89)	65.2% (58/89)	62.9% (56/89)
pass$\hat{}$3 / pass$\hat{}$5 (all k attempts pass)	42 / 34%	30 / 22%	22 / 15%	7 / 2%	6 / 2%
bands easy / in-band / hard	30 / 42 / 17	20 / 42 / 27	13 / 53 / 23	2 / 56 / 31	2 / 54 / 33
runnable app produced	100.0%	99.8%	100.0%	99.8%	69.0%
Tier A / B / C pass rate	96 / 68 / 93%	89 / 61 / 89%	91 / 53 / 84%	87 / 44 / 65%	61 / 38 / 48%
over-permissive : over-strict (first failure)	10 : 626	6 : 642	29 : 1532	65 : 1313	6 : 958
over-permissive : over-strict (all probes)	24 : 1609	19 : 1329	43 : 4208	86 : 3305	31 : 2575

D SECTOR ROLLUP

We report domains at two named levels. At the task level there are 89 distinct business settings and no setting is repeated. Table 5 gives the corpus-wide sector rollup, generated by the same script that produces every figure.

Table 5: Sector rollup of the 89 task-level settings.

Sector	tasks (of 89)
Finance / accounting / securities	20
Government / municipal / elections	17
Media / events / hospitality / consumer	14
Logistics / trade / transport	10
Agriculture / food / energy and utilities	10
Manufacturing / industrial QA	8
Healthcare / lab operations	7
Insurance / warranty	3

E FAILURE-FAMILY TAXONOMY

Table 6 lists the 17 designed failure families with a definition, the task count, the pooled pass rate of the five models on the family's tasks, and one example task. The orphan-reject family was deliberately downsampled from 35% to 14% of the corpus at curation. Effective diversity by perplexity is 13.2 families.

Table 6: Designed failure families with pooled five-model pass rate and one example task per family. Example task names are release identifiers.

Family	tasks	pooled pass rate	Definition (what a failing design does)	Example task
Orphan / referential integrity	13	52%	accepts a write naming a non-existent parent	`commercial_rent_abatement`
Effective dating / status	10	38%	wrong as-of semantics; status not derived from dated records	`cmrs_discontinuance`
Stated rule not enforced	9	50%	a stated domain rule is not enforced at write time	`allergen_line_schedule`
Derived, not stored	9	39%	caches a value the specification requires to be re-derived; stale after a correction	`chargeback_freeze`
Business-key uniqueness	8	56%	business identity not backed by a database unique key	`building_inspections`
Over-denormalization	8	43%	stores a derivable fact redundantly (structure-scan catch)	`custom_feedyard`
Append-only / stale read	7	30%	edits history in place; reads miss superseding entries	`batch_record_503b`
Immutability after sign-off	5	59%	signed records remain mutable; captured figures recomputed live	`assessment_roll`
Value-shape (exactly-one)	5	25%	both-or-neither accepted where exactly one is required	`buyin_desk`
Empty set vs zero row	3	34%	empty result where a zero-valued row is required, or the reverse	`coop_estimated_billing`
Interval edge semantics	3	13%	boundary day included or excluded wrongly; degenerate ranges mishandled	`elevator_service_contract`
Sign / magnitude	2	50%	inverted sign turns a shortfall into a credit	`ad_sales_shortrate`
Idempotency / upsert	2	34%	an upsert swallows a violation the specification says to reject	`ata_carnet`
Boolean vs timestamp read	2	74%	raw timestamp projected where the derived boolean is specified	`deferred_defect_desk`
Proration / rate calendar	1	4%	wrong proration base across a rate boundary	`banquet_beo`
Retroactive effectivity	1	24%	a late-arriving fact does not reflow derived history	`bus_garage_timekeeping`
Other	1	28%	task-specific	`grain_elevator`

F WORKED EXAMPLE: ROYALTY NETTING

The brief for `royalty_netting` is 38 lines. Its operative paragraphs read, verbatim:

> We pay royalty owners, and depending on their lease we net out post-production costs before figuring what they're owed. The lease language is the whole story here. Some leases let us deduct a proportionate share of gathering, compression, dehydration, and marketing costs; some are cost-free and we deduct nothing. It's never a company-wide decision. One lease can bear a cost category while the lease right next to it does not, even for the same product and month.
>
> Each owner goes in once, with their owner number, name, lease number, and their decimal interest (a fraction like 0.012500). [...] the same owner number never gets entered twice.
>
> For each owner I record, per product, month, and cost category, whether they bear that category. If there's no entry for a given owner, product, month, and category, treat that as not borne. Absent means not borne, plain and simple.
>
> [...] An owner's deduction for a category is their decimal interest times that category's pool total, but only if their terms say they bear it that month. Otherwise it's zero. Their net for the month is the sum across all categories. Always figure this fresh off the current pool total. Don't freeze it.
>
> Corrections happen. When an accountant corrects a prior month's invoice, everybody affected recomputes off the corrected amount. The correction always governs that product, month, and category going forward, whether or not an original invoice was ever recorded, and the latest correction on file is the one that counts. Keep both the original and the correction so I can show what governed each month. Correction references share the same namespace as invoice references, so no reference can collide with any other.

The brief is followed by a build appendix that names the platform, requires every identity and immutability rule to be a real database constraint, and requires the manifest and a headless self-test; the contract (`company_standards.md`) fixes the operations, their argument names and wire formats, and the interface conventions.

Graded properties. Tier B replays twelve hidden workflows against the built application: one happy path (owners, terms, invoices, then netting reads), three rejection workflows (a duplicate owner number, a duplicate invoice reference, a correction reference that collides with an invoice reference), one acceptance workflow (a correction to an existing invoice is accepted), and seven anomaly workflows, each aimed at one sentence of the brief: deductibility differs per lease for the same product and month; a share is re-derived after a term changes; a correction re-derives every affected net; the latest of two corrections governs; a correction with no original invoice establishes the pool; an absent term means not borne; and the net sums only the borne categories. Tier A inserts a duplicate owner number and a colliding reference by raw SQL and requires the database to refuse both, and the structure scan checks that no netted figure is stored. Tier C signs in and checks that the netting panel renders the owner rows the backend returned.

Difficulty. The workflow rules are ordinary business logic with sharp edges. "The latest correction on file is the one that counts" and "whether or not an original invoice was ever recorded" together mean the pool total is a function of the correction history and not of the invoice table, and a model that resolves it by joining to the original invoice returns nothing for a stray correction. "Absent means not borne" is an empty-set-versus-zero contract that an inner join silently violates. The database rules are equally concrete: a reference namespace shared by two record types cannot be enforced by two per-table unique keys, so a correct design needs a single references table or a trigger. None of these is exotic, and all of them are stated. Across the 25 attempts, 3 passed (Opus 4.8 twice, DeepSeek V4 once). Of the 22 losing attempts, 11 rejected the happy path's first write because the workload sent the month as a date, as the contract specifies, and the application demanded `YYYY-MM` (nine attempts) or some other unstated condition (two); 7 returned a wrong netting figure on a read; 3 Qwen attempts failed the database probes; 1 returned a malformed response. The model that most often reasoned the netting correctly, GPT-5.6 Sol, went 0/5, four times on the month format.

G PUBLIC/PRIVATE PARTITION

Following the paired public/private-leaderboard practice of GAIA and SWE-Bench Pro (Mialon et al., 2024; Scale AI (SEAL Labs), 2025), nine tasks form the public set and eighty the private set;

Table 7 lists which artifacts are released for each. Publishing gold variants and workloads would make the public set unusable as a durable evaluation, so we withhold them and instead release the grader, the protocol, the agent harness, and per-attempt grade records for the public tasks. Splits for adaptation experiments are family-held-out: the in-distribution (ID) split holds out instances of failure families seen in adaptation data, the out-of-distribution (OOD) split holds out entire families, and the gap between them is the transfer signal. We do not split by domain, which is uncorrelated with skill here and would leak it; the private set is family-stratified so both splits survive the partition. Others score against the private set by submitting an agent harness in the released task format; the authors run it under the configuration of Appendix H and report the metrics of Appendix C.

Table 7: Public/private partition.

Artifact	Public set	Private set
Brief + contract (what the model sees)	✓	withheld entirely
Deterministic grader + wiring-check test suite	✓	same code
Agent harness with the five reference configurations	✓	same code
Gold workloads / probes / seeds	withheld	withheld
Gold variants	withheld	withheld
Reference-model trajectories, built applications, grade records	✓	–
Task count	9	80 (family-stratified)

H Evaluation Configuration

Table 8 lists the endpoint, reasoning setting, and budget used for every model.

Table 8: Evaluation configuration. All runs use OpenHands CLI 0.62.0 with native function calling, 200 steps and a 4-hour wall per attempt, one fresh sandbox per attempt, and one grading pass per attempt. Cost is a token-based estimate at list price; the Opus 4.8 run is not itemized.

Model	Endpoint	Reasoning setting	Request timeout	Notes	Est. cost
Claude Opus 4.8	Anthropic API	maximum	2,100 s	prompt caching on the system prompt and the last two turns	–
Kimi K3	DashScope	effort high	2,100 s	single turns of 11–32k output tokens need the long request timeout	$815
GPT-5.6 Sol	OpenAI Responses API	effort max	2,100 s	128k output cap, 1.05M context; Chat Completions caps this model at xhigh	$1,070
DeepSeek V4 Pro	DashScope	effort high	720 s		$42
Qwen3.8-Max	DashScope	thinking budget 4,000	720 s	no “high” setting exposed	$487

Harness fixes required for open models. OpenHands disables tool calling for model names it does not recognize, which silently breaks Kimi; we force native function calling on. DashScope rejects `top_p`, which is dropped, and OpenHands 0.62 hard-rejects a tool call that omits its `security_risk` argument, which we backfill. Software-development-kit retries are disabled so that any real hang is a visible timeout handled by the harness’s own retry loop.

A harness property that affected most attempts. The command runtime of OpenHands 0.62 reads a command’s standard output to end of file after the command exits. When an agent starts its server in the background from a shell command (the pattern `bash run.sh &`), the server inherits the pipe, the read never returns, and the agent’s turn blocks until the wall. A census of the trajectories finds this in 90% of Opus attempts (median 33.6 minutes of wall lost after the blocking command), 96% of DeepSeek attempts (20.3 minutes), 83% of Sol attempts (19.1 minutes), 47% of Qwen attempts (7.5 minutes), and 23% of Kimi attempts (4.7 minutes). Pass rates of affected and unaffected attempts are similar within each model (for Opus, 43% against 52%; for Kimi, 37% against 36%), consistent with the block arriving after the application was built. The property is part of the configuration under which every number in this paper was measured; the released harness exposes an opt-in drain timeout, off by default, so that future runs can measure its cost directly.

I PER-TASK RESULTS

Table 9 gives the pass count (passes / attempts) for every task and model, sorted by total pass count. The eight tasks solved by no model are `batch_record_503b`, `buyin_desk`, `carlton_club_membership`, `country_elevator`, `dividend_entitlement`, `elevator_service_contract`, `two_rivers_festival`, and `waste_accumulation`; each has a passing oracle implementation and 25 failing verdicts with per-probe detail in the release.

Table 9: Pass count (passes / attempts) per task and model, sorted by total pass count over the five models.

Task	Family	Sector	Opus	Sol	Kimi	DS	Qwen
fishing_overage_carryover	Business-key uniqueness	Government	4/4	4/5	5/5	4/5	3/5
chargeback_freeze	Derived, not stored	Finance	4/5	5/5	4/5	4/5	2/5
customs_export_declaration	Stated rule not enforced	Logistics	4/4	5/5	2/5	5/5	3/5
deferred_defect_desk	Boolean vs timestamp read	Media	5/5	5/5	3/5	3/5	3/5
facultative_reinsurance	Orphan / referential integrity	Insurance	4/5	4/5	4/5	4/5	3/5
voter_rolls	Over-denormalization	Government	4/4	5/5	5/5	2/5	3/5
contractor_ledger	Append-only / stale read	Finance	5/5	4/5	5/5	2/5	2/5
dairy_producer_ledger	Derived, not stored	Agriculture	4/4	5/5	3/5	3/5	3/5
drug_billable_units	Orphan / referential integrity	Healthcare	4/5	4/5	5/5	2/5	3/5
flat_fee_billing	Orphan / referential integrity	Finance	3/3	4/5	5/5	1/5	5/5
ftz_inventory	Over-denormalization	Logistics	3/4	5/5	3/5	3/5	4/5
loan_payment_waterfall	Derived, not stored	Finance	5/5	4/5	2/5	3/5	4/5
gym_turnstile	Boolean vs timestamp read	Media	4/4	5/5	3/5	3/5	2/5
lottery_retailer_reconciliation	Orphan / referential integrity	Finance	5/5	5/5	5/5	0/5	2/5
rate_circular	Effective dating / status	Logistics	5/5	5/5	3/5	1/5	3/5
allergen_line_schedule	Stated rule not enforced	Agriculture	4/4	5/5	2/5	4/5	1/5
county_ledger	Business-key uniqueness	Finance	5/5	1/5	4/5	3/5	3/5
fleet_endorsements	Over-denormalization	Logistics	2/5	5/5	5/5	1/5	3/5
foul_time_ledger	Stated rule not enforced	Logistics	4/5	3/5	3/5	3/5	3/5
permanency_hearing_calendar	Business-key uniqueness	Government	2/3	5/5	2/5	4/5	3/5
same_day_ballot	Orphan / referential integrity	Government	5/5	4/5	5/5	1/5	1/5
cryostorage_straws	Business-key uniqueness	Healthcare	4/5	4/5	3/5	2/5	2/5
estate_auction	Orphan / referential integrity	Finance	5/5	4/5	3/5	1/5	2/5
leak_adjustment	Sign / magnitude	Agriculture	4/5	3/5	2/5	1/5	5/5
tpr_filing_window	Stated rule not enforced	Government	3/5	5/5	5/5	0/5	2/5
transfer_agency	Immutability after sign-off	Finance	2/4	3/5	5/5	2/5	3/5
warranty_claim_resolution	Business-key uniqueness	Insurance	3/4	3/5	5/5	2/5	2/5
assessment_roll	Immutability after sign-off	Government	4/4	2/5	4/5	1/5	3/5
building_inspections	Business-key uniqueness	Government	3/5	4/5	5/5	1/5	1/5
lot_disposition	Over-denormalization	Media	5/5	1/5	3/5	3/5	2/5
merchant_reserve	Derived, not stored	Finance	5/5	4/5	2/5	2/5	1/5
pretreatment_surcharge	Orphan / referential integrity	Government	4/4	5/5	3/5	0/5	2/5
settlement_statement	Immutability after sign-off	Finance	4/4	3/5	3/5	2/5	2/5
borescope_inspection	Append-only / stale read	Manufacturing	2/3	5/5	2/5	3/5	1/5
bridge_inspection	Immutability after sign-off	Government	4/4	2/5	2/5	4/5	1/5
crestview_holds	Effective dating / status	Media	3/5	5/5	2/5	0/5	3/5
fee_cost_recovery	Orphan / referential integrity	Finance	2/4	1/5	2/5	4/5	4/5
mdr_freeze	Immutability after sign-off	Healthcare	4/4	2/5	4/5	2/5	1/5
mill_heat_cert	Orphan / referential integrity	Manufacturing	4/4	1/5	2/5	3/5	2/5
open_loan_book	Stated rule not enforced	Finance	5/5	1/5	4/5	0/5	2/5
copacker_lots	Stated rule not enforced	Manufacturing	2/4	5/5	1/5	3/5	0/5
dealer_trade	Business-key uniqueness	Finance	4/4	2/5	4/5	0/5	1/5
infusion_charge_capture	Value-shape (exactly-one)	Healthcare	4/5	5/5	2/5	0/5	0/5
mold_housing	Value-shape (exactly-one)	Government	3/3	2/5	4/5	1/5	1/5
shop_genealogy	Empty set vs zero row	Manufacturing	2/4	1/5	4/5	4/5	0/5
ad_sales_shortrate	Sign / magnitude	Media	4/5	5/5	0/5	1/5	0/5
ata_carnet	Idempotency / upsert	Logistics	4/4	0/5	3/5	2/5	1/5
cmrs_discontinuance	Effective dating / status	Government	3/4	0/5	4/5	2/5	1/5
commercial_rent_abatement	Orphan / referential integrity	Media	4/5	2/5	1/5	3/5	0/5
concrete_cylinder_lab	Append-only / stale read	Manufacturing	0/4	5/5	3/5	2/5	0/5
dot_inspection_compliance	Effective dating / status	Logistics	3/3	4/5	1/5	1/5	1/5
poll_worker_staffing	Interval edge semantics	Government	0/4	3/5	4/5	0/5	3/5
specimen_volume_ledger	Derived, not stored	Healthcare	4/4	4/5	1/5	1/5	0/5
tollfree_lifecycle	Stated rule not enforced	Government	3/4	4/5	2/5	1/5	0/5
patent_renewals	Effective dating / status	Finance	4/4	2/5	2/5	1/5	0/5
propane_keepfull	Effective dating / status	Agriculture	4/5	0/5	5/5	0/5	0/5
virtual_queue	Effective dating / status	Media	4/5	0/5	3/5	1/5	1/5
royalty_netting	Empty set vs zero row	Finance	3/4	0/5	0/5	5/5	0/5
grain_elevator	Other	Agriculture	3/5	0/5	3/5	1/5	0/5
loaner_kit_cage	Stated rule not enforced	Media	1/5	3/5	1/5	1/5	1/5
ticketing	Over-denormalization	Media	3/4	0/5	1/5	0/5	3/5
tool_crib_gauge_custody	Value-shape (exactly-one)	Manufacturing	2/5	4/5	1/5	0/5	0/5
bus_garage_timekeeping	Retroactive effectivity	Logistics	3/5	0/5	1/5	2/5	0/5
commercial_policy_endorsements	Effective dating / status	Insurance	2/5	0/5	1/5	2/5	1/5
factory_incentives	Orphan / referential integrity	Manufacturing	3/4	2/5	0/5	0/5	1/5

Table 9 continued

Task	Family	Sector	Opus	Sol	Kimi	DS	Qwen
interline_car_hire	Append-only / stale read	Logistics	3/5	3/5	0/5	0/5	0/5
outofwork_book	Idempotency / upsert	Government	0/4	0/5	4/5	1/5	1/5
zoning_variance	Orphan / referential integrity	Government	2/3	3/5	0/5	1/5	0/5
coop_estimated_billing	Empty set vs zero row	Agriculture	1/5	3/5	0/5	1/5	0/5
spirits_production_close	Effective dating / status	Agriculture	3/4	0/5	1/5	0/5	1/5
custom_feedyard	Over-denormalization	Agriculture	0/5	2/5	0/5	0/5	2/5
warehouse_replenishment	Over-denormalization	Logistics	2/3	2/5	0/5	0/5	0/5
film_rights_library	Append-only / stale read	Finance	0/4	0/5	0/5	3/5	0/5
soccer_registry	Derived, not stored	Government	2/4	0/5	0/5	0/5	1/5
cemetery_endowment	Stated rule not enforced	Government	1/3	0/5	1/5	0/5	0/5
banquet_beo	Proration / rate calendar	Media	0/4	0/5	0/5	0/5	1/5
coop_net_metering_bills	Derived, not stored	Agriculture	0/5	0/5	1/5	0/5	0/5
lab_standards	Effective dating / status	Healthcare	1/4	0/5	0/5	0/5	0/5
pension_election	Append-only / stale read	Finance	0/3	1/5	0/5	0/5	0/5
rent_roll	Derived, not stored	Finance	1/4	0/5	0/5	0/5	0/5
vacation_points_club	Derived, not stored	Media	0/5	0/5	0/5	1/5	0/5
batch_record_503b	Append-only / stale read	Healthcare	0/4	0/5	0/5	0/5	0/5
buyin_desk	Value-shape (exactly-one)	Finance	0/5	0/5	0/5	0/5	0/5
carlton_club_membership	Business-key uniqueness	Media	0/5	0/5	0/5	0/5	0/5
country_elevator	Value-shape (exactly-one)	Agriculture	0/4	0/5	0/5	0/5	0/5
dividend_entitlement	Over-denormalization	Finance	0/4	0/5	0/5	0/5	0/5
elevator_service_contract	Interval edge semantics	Media	0/3	0/5	0/5	0/5	0/5
two_rivers_festival	Orphan / referential integrity	Media	0/5	0/5	0/5	0/5	0/5
waste_accumulation	Interval edge semantics	Manufacturing	0/4	0/5	0/5	0/5	0/5

J Failure Signature Detail

Over-strictness is the dominant failure for every model. The generation literature predicts the opposite of what we observe. Over 90% of robustness deficiencies in generated code are missing conditional checks (Li et al., 2025), security audits report vulnerability rates dominated by missing input validation (Pearce et al., 2022), negation is handled markedly worse than affirmation (García-Ferrero et al., 2023), and shortcut learning (Geirhos et al., 2020) and sycophancy (Sharma et al., 2024) push toward the happy path; over-strictness has been treated as the rare inverse error (Krakovna et al., 2020). Our data reverses that expectation for every model we tested. Rejecting a contract-valid write is the most common first failing check for four of the five models (39% of Opus's losing attempts, 50% of Kimi's, 61% of Sol's, 31% of DeepSeek's) and the second most common for Qwen (33%), behind failing to produce a runnable application. Counted per workflow, over-strict Tier-B failures outnumber over-permissive ones by 87 to one for Opus, 64 to one for Kimi, 266 to one for Sol, 28 to one for DeepSeek, and 323 to one for Qwen. The forensic audit describes what the strictness looks like in code: whole-number gates on money, two-decimal regular expressions, `YYYY-MM` month formats where the contract sends a date, strict ISO 8601 (International Organization for Standardization date-time format) parsing, and `typeof`-string checks on contract-typed numbers, each rejecting a workload value the contract licenses before the data-model logic runs. The generation literature predicts permissive failure (§2); here the direction is reversed for every model, and the size of the ratio is a per-model trait.

Wrong reads are the second failure, and the database tier is where models differ least. A wrong read result (a derived value, a row set, or an empty-set-versus-zero-row contract) is the first failing check for 32% of Opus's losing attempts, 29% of DeepSeek's, 23% of Kimi's, 15% of Sol's, and 12% of Qwen's. Tier-A failures are a minority everywhere (11–21% of losing attempts), and within them mutable signed figures outnumber application-code-only uniqueness for every model except Sol, whose Tier-A losses are led by uniqueness. Interface wiring is a first failure for 12% of Opus's and 11% of DeepSeek's losing attempts and for at most 6% of the others'.

The remainder of this appendix decomposes the 1,289 losing attempts by first failing check, in tier order, and records what a source-level audit of the built applications found behind each category. Shares are of all losing attempts.

Over-strict writes (38%). Prior work on generated code reports the opposite failure, missing checks rather than extra ones (Li et al., 2025; Pearce et al., 2022), and treats over-strictness as the rare inverse error (Krakovna et al., 2020). Our data reverses that expectation. Rejecting a contract-valid write is the most common first failing check for four of the five models (43% of Opus's losing attempts, 47% of Sol's, 49% of Kimi's, 32% of DeepSeek's) and the second most common for

Qwen (24%). Counted per workflow, over-strict Tier-B failures outnumber over-permissive ones by 53 to one for Opus, 107 to one for Sol, 53 to one for Kimi, 20 to one for DeepSeek, and 160 to one for Qwen (Figure 4b); the over-permissive count (45 first failures, 3%) is a lower bound, since a rejection probe whose own setup write was refused is credited as enforced. The largest class of rejection message (96 of 484) is rule-shaped: the application encodes a business rule that the brief states or implies, and that the contract never operationalizes, as a 409, and the gold workload then performs the action the sentence seemed to forbid. On cemetery endowment the brief opens with "we cannot sell a plot in a garden until that garden's care fund has been pre-funded to its legal minimum"; the gold implementation sells one while its own eligibility read returns false, and 13 of the 15 rejections at that step name the rule. The smaller classes are wire shape (84; a month sent as a full date or a timestamp in Postgres text form, §5), number shape (68; whole-number checks on quantities the brief names as counted things and the workload sends as fractions), login demanded on a contract-open route (44), and a vocabulary not in the application's list (37).

Wrong reads (21%). A wrong read result is the first failing check for 270 attempts, the second-largest loss for every model except Qwen. Almost all are semantic rather than numeric: a looser money comparator would recover at most three. The rest read the same records differently: which wage governs a back-dated week, whether a mid-month accrual folds into its month, what a reissue does to the old ticket's coupons. On ticketing, three of five models mark the old coupons "exchanged" after a reissue, because the brief lists that state and no operation in the contract produces it; the gold implementation leaves them open. There is no partial credit for a consistent alternative reading.

Database enforcement (15%). Tier-A failures are the first failing check for 193 attempts, 10–21% of each model's losses. Uniqueness enforced only in application code leads (97), but a source-level check splits them: 16 declare a key the schema does not enforce, 34 disclose a nominal key for an operation the brief describes as append-only, and about 47 enforce the key in a form the probe's no-row fallback cannot see (§5). Sol's 40 are all of the last two kinds, a schema style rather than a data-model error. Signed figures recomputed on read rather than stored account for 63, half of them after a rejected setup write, and keys enforced but never disclosed for 26, 25 of them DeepSeek's.

No runnable application (11%). 148 attempts end with nothing to boot, 142 of them Qwen's, and fail every tier at once. The cross-harness comparison of §4.4 shows this loss belongs largely to the agent harness rather than the model.

Interface wiring (12%). Tier-C failures are the first failing check for 149 attempts: 131 panels that do not render the rows the backend returned and 18 refusals the interface does not surface. The share is 17% of Opus's losing attempts, 13% of DeepSeek's, and 9–11% of the others'.

Wire conventions in the workload. The workload sends month-typed arguments as full dates (`month: Ï2026-01-21¨`, `effective_month: Ï2026-03-01¨`) and some timestamp arguments in Postgres text form (`Ï2026-01-02 09:00:00¨`). The contract fixes argument names and no wire shape; its one sentence about dates concerns interface output. An application that validates a month as `YYYY-MM` or a timestamp as ISO 8601 rejects the write: 30 first failures on four tasks name the month form and 24 on nine tasks name a date or timestamp format. Accepting the string and casting it is the safer choice, and the applications that pass do so, but the loss is decided by an undisclosed convention rather than by the data model. We state the convention here and leave the workload as shipped.

Limits of the uniqueness probe. When no row exists to drive a duplicate, Tier A confirms a declared key by name against the catalog and, failing that, against the index and constraint definition text. Keys realized as a trigger-assigned sequence number, a composite that includes an identity column, a nullable column, or an EXCLUDE or partial constraint can be enforced and still read as missing; about 47 of the 97 uniqueness losses in §4.3 are of this kind, and the verdicts stand as graded. A probe that drives the duplicate through the application's own write path would remove the class.

K Cross-Harness Comparison

Because the coding-agent harness is part of the system under test (§4.1), a natural question is how much of a model's score is the model versus the harness. On a stratified random subset of 20 of the

89 tasks (sampled by consensus difficulty band, seed fixed and published), we rebuild all five models under a different agent, the OpenCode CLI, holding everything else fixed: the same 200-step, 4-hour budget, per-model endpoints and reasoning settings (Appendix H), a fresh sandbox and workload seed per attempt, and the same frozen grader. Table 1 (§4.4) pairs the two harnesses within task; the OpenHands column is the paper's own attempts restricted to these 20 tasks.

The comparison is stable where it matters and informative where it moves. The model ordering is identical under both harnesses: Opus leads, the middle pair and the bottom pair keep their order, and OpenCode is equal-to-slightly-better for every model. Four of the five paired differences sit inside their 95% interval and are ties (Opus $+4$, Kimi $+3$, Sol $+7$, DeepSeek $+1$ percentage points). The single exception is Qwen3.8-Max at $+18$ points (better on 12 of 20 tasks, sign-test $p = 0.077$), and its cause is reliability rather than reasoning: under OpenHands, Qwen produced no runnable application on 28% of these attempts, and under OpenCode on none. Across all five models OpenCode yielded a runnable application on all but four attempts (four Kimi builds), against the no-app tax OpenHands levies on the weakest model. The separation the paper attributes to the business-logic tier therefore survives a change of harness; what a better-instrumented harness recovers is chiefly the attempts a weaker model would otherwise lose before its data-model logic is ever graded. One caveat: the DeepSeek subset absorbed a provider rate-limit episode that consumed part of its wall on several attempts, so its 36% is a mild undercount, which does not change the ordering.

L GRADING-ARTIFACT TAXONOMY

Table 10 gives the artifact classes found in the pilot audit of §5, with the mechanism and the fix that entered the frozen grader.

Table 10: Full-stack grading-artifact taxonomy: comparison, dialect, or bookkeeping effects that can fail a correct application. Identifying these required adversarial review of real agent-built applications against the grader; we report them so others building execution-graded application benchmarks can pre-empt them.

Artifact class	Mechanism	Fix
Rendering-layer value re-check (earlier Tier C)	interface gate compared exact values through the rendering layer; commas, null dashes, labeled enums, and extra columns read as "not wired"; the grader's own replay left typed inputs unfilled	wiring-only anchor evidence (§3.2); per-type input coercion in the driver
Undisclosed grader dialect	gold arguments serialized in forms the contract never states: scientific notation (`1E+2`; 47 tasks, 1,110 values), language-specific list serializations (1 task, unwinnable), canonical enum tokens graded but never stated, login request shape unspecified	value-identical gold normalization; contract disclosure clauses; grader sends `Accept: application/json` and honors redirect cookies
Comparison intolerance	cents-rounded gold against a correct unrounded fold at 10^{-4} equality	strict-first compare, cent-rounding retry gated to $\lvert n\rvert \geq 0.1$ (rates keep full precision; off by a cent still fails)
Verdict bookkeeping	stored pass contradicting the three tier booleans	pass recomputed from tier booleans as the single source of truth

M LIMITATIONS AND OPEN PROBLEMS

Limitations. (1) *Reference-relative measurement.* Bands and failure signatures are properties of specific models under a specific harness, so downstream users must re-band per configuration. (2) *Trap masking.* Upstream defects, chiefly over-strictness, pre-empt designed traps, so headline scores understate per-model data-model coverage; read them with the first-failure decomposition (§4.3). (3) *Property checks rather than proofs.* The checks catch the defects they target, not semantic equivalence in general, and the redundancy scan can be sensitive to column naming on one audited task. (4) *Dialect recurrence.* Until the derivability lint gates every run, new artifact classes

can recur; only the Opus attempts had a code-level audit, so an artifact specific to another model's style could remain undetected.

Open problems. The cross-harness comparison of §4.4 runs one rung of a longer ladder; extending it to more agents at matched budgets would separate model capability from harness affordance, and we release the harness on a standard task format to make that cheap. A post-training study that adapts on the public set and evaluates on held-out failure and product families would test whether invariant reasoning generalizes, and product-isomorph families, in which the same invariants are stated with less explicit contracts, would add the implied-specification axis discussed in §6.