BhashaBench-Agentic: A Multilingual, Multi-Turn Benchmark for Agentic Tool Use

Anonymous authors
Paper under double-blind review

Abstract

BhashaBench-Multi established a verified, multilingual, multiple-choice benchmark for Indian-languages covering domain knowledge across Agriculture, Finance, Law, and Ayurveda. BhashaBench-Agentic asks a different, complementary question about the same domain content: not whether a model knows a fact, but whether it can *act* on it inside a multi-turn conversation – asking when a required detail is missing, confirming before changing anything, calling the correct tool with the correct arguments, and grounding its answer in what a tool actually returned. This paper describes the BhashaBench-Agentic's design: a six-action turn loop, a ten-code trajectory taxonomy, and a four-signal scoring rule. An episode counts as **solved** only when all four signals pass together, not on average (a plain mean of the same four signals reads 0.895 against the strict rule's 0.215; we report both, as a sensitivity check, not a resolution) – so a deployment cannot trade a safety failure on one turn for a fluent answer elsewhere. The corpus is built from four data-curation pipelines – hand-authored, persona-conditioned, mined from BhashaBench-Multi, and mined from de-identified Supreme Court judgments – and checked with build-time validity gates, cross-language parity tests, and an ∼55,000-case mutation and golden-replay test suite. The resulting corpus has 4,258 scored episodes across 82 seed scenarios (goals with starting state and tools) and 19 languages. We report results for twelve models (§7), and use two of them, gpt-oss-120b and Qwen3-235B-A22B, for a detailed case study (§8): checking a specific low-score pattern against the episode's own ground truth, and finding it consistent with the model treating recognition of a real case as sufficient without verifying it, which produces a fabricated citation. A disclosed, minimal prompt fix measurably improves this for both models, confirmed at full corpus scale. We are explicit throughout about what our evidence does and does not establish.

1 Introduction

BhashaBench-Multi (Devane et al., 2025; BharatGen, 2025) is a verified, multiple-choice benchmark on 22 scheduled Indian languages covering knowledge from 4 domains: given a question in Agriculture, Finance, Law, or Ayurveda, does a model selects the factually correct answer, in the language it was asked in. BhashaBench-Multi is a necessary and, by itself, insufficient test of what a deployed assistant needs to do. A model can correctly answer "what interest rate applies to a delayed tax refund" and still fail the person in front of it by inventing a document that does not exist, by hallucinating a number instead of asking for it, or by changing an account's state before the user has agreed to it. Multiple-choice evaluation has no turns, no tools, and no state, so it structurally cannot exercise any of this – not because the format is weak, but because the questions' set built to ask (*does the model know X*) is different from the one an agentic deployment needs to answer (*does the model act correctly on X, across a conversation, when something is missing or ambiguous*), a distinction tool-use benchmarks generally are built around (Liu et al., 2023; Yao et al., 2024).

BhashaBench-Agentic is built to ask that second set of questions, using the same domain content, which is already verified by BhashaBench-Multi. It reuses BhashaBench-Multi's questions and facts as a source of grounded native-quality material (§5), but authors the

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.

agentic structure – tools, multi-turn scripts, the exact call sequence a correct agent should make, the state those calls read and write – independently. The two benchmarks are complementary, not competing.

This paper covers BhashaBench-Agentic's mechanics and four data-curation pipelines (§2–§5); seed-level and source-level evidence that it measures what it claims to (§6); baseline results for twelve models on the same 4,258-episode corpus, two of them also re-run with a disclosed prompt intervention (§7); and one worked example of using the four signals to trace a low score to its cause – checking a specific claim against ground truth rather than reporting it on trust (§8).

This paper's two contributions are: (1) BhashaBench-Agentic itself – a 4,258-episode, 19-language, four-signal agentic benchmark built from four independently-verified data pipelines, with build-time validity gates and an 8-mutation scorer-verification suite that together check the BhashaBench-Agentic measures what it claims to; and (2) a demonstration, using BhashaBench-Agentic's own instrumentation, that a low aggregate score can be traced to a specific, ground-truth-verifiable mechanism – recognition substituting for verification – rather than reported as an opaque number, together with a minimal, disclosed prompt intervention that measurably addresses it.

2 Terminology

We define BhashaBench-Agentic's core vocabulary here, in plain terms, before using it below.

- **Seed scenario.** A user goal, a starting sandbox state (e.g. which documents exist in a legal matter), and the subset of tools available. Written or mined once; the rest follows from it.
- **Episode.** One complete, scored conversation: a specific seed scenario, in a specific language and script, following a specific trajectory (below). The unit that gets a score.
- **Trajectory.** The *shape* of the conversation for a given seed – smooth (*T0*), requiring a clarifying question (*T3*), a tool failure to recover from (*T6*), and so on (§4.3 lists all ten codes).
- **Harness.** The program that runs an episode: presents user turns, accepts the model's action, executes tool calls against a per-episode sandbox – the agent's *environment*, whose state the harness returns as the model's next observation – and logs everything for scoring, unmodified for every model we report on (architecture in Fig. 4, Appendix B).
- **Gold trace.** The reference solution for an episode: the tool calls a correct agent should make and the end state or answer expected. Authored once per seed, shared unchanged across every language variant (§4.1).
- **Action.** One of six choices in the benchmark's *action space*, chosen fresh each turn: *clarify* (ask for a missing detail), *ir* (declare the intended tool call, a plan, before executing), *confirm* (state intent and ask approval), *call* (execute a tool), *respond* (final answer), or *halt* (stop without acting). Declaring an `ir` is not the same as issuing its `call` – §8 relies on this.
- **Signal.** One of four separately-scored per-episode measures (*execution*, *process*, *call-match*, *policy*; §4.4) that together determine whether an episode is *solved*.

3 Related Work

ReAct (Yao et al., 2022) and Toolformer (Schick et al., 2023) established interleaved reasoning-and-acting – an explicit agent-environment loop over an action space of tool calls – as the dominant paradigm for language-model tool use. AgentBench (Liu et al., 2023), ToolLLM (Qin et al., 2023), and the Berkeley Function-Calling Leaderboard (Yan et al., 2024; Patil et al., 2023) evaluate this at scale, mostly in English, on single- or lightly multi-turn calls.

τ-bench (Yao et al., 2024) and its successor τ^2-bench (Barres et al., 2025) evaluate a model acting as a customer-service agent across a full multi-turn conversation against a real API and database, and score an episode solved only when every applicable check passes – a product of individual checks' scores, not a mean, since a mean can hide exactly the failure a deployment cares about. This is the closest prior scoring philosophy to ours (§4.4), but two

things neither they nor the tool-use benchmarks above test are what motivate BhashaBench-Agentic's two additional signals. First, none of them score a standing, cross-turn rule that a model must remember and recognize as applying on its own, independent of any single task's instructions; BhashaBench-Agentic adds an explicit *policy* signal for exactly this (§4.4). Second, none of them make a model's own respect for a pre-action confirmation step measurable when the harness itself does not enforce that step; BhashaBench-Agentic's confirm gate is observed by the harness but not enforced (§4.5), so whether a model respects the gate is itself part of what gets measured, rather than guaranteed by construction. Most agentic benchmarks, including the ones above, are also English-only or treat other languages as a translation pass; BhashaBench-Agentic instead authors one gold trace per seed that never fixes a language-specific string where a language-specific answer is unavoidable (§4.1) and verifies, as a build-time test, that no target-language text appears inside it, so a cross-language score gap is attributable to the model rather than to how the reference was authored.

On the language side, prior Indic-language benchmarks are almost entirely static and non-agentic. BhashaBench-Multi (Devane et al., 2025; BharatGen, 2025) verifies multiple-choice domain knowledge across the same four domains BhashaBench-Agentic draws content from. IndicGenBench (Singh et al., 2024) evaluates generation tasks – summarization, translation, and question answering – across 29 Indic languages, and MILU (Verma et al., 2025) evaluates broad multi-task knowledge, including culturally specific content, across 11 Indic languages. All three, like the agentic benchmarks above, score a single-turn response against a fixed reference; none place a model inside a multi-turn conversation with tools, state, and a confirm gate, which is the gap BhashaBench-Agentic is built to fill using their kind of domain content as source material (§1).

4 Benchmark Design

This design is what makes the benchmark *agentic* rather than a static question-answering test (§1): a model completes an episode only by choosing among six actions turn by turn (§4.1–§4.2), calling real tools against a stateful sandbox, and observing what those tools return before its next choice – the perceive-act-observe loop a one-shot answer cannot exercise. It is also why a confirm gate the model is free to violate (§4.5) is itself part of what gets measured: testing whether an agent chooses not to act without permission presupposes a system that can act on the world in the first place.

4.1 Episode structure

An episode is a triple $E = (s, (\ell, c), t)$: a seed scenario s, a language/script pair (ℓ, c) (e.g. Hindi in Devanagari or in Roman transliteration), and a trajectory code t (§4.3). This is also the multiplier that turns 82 authored or mined seeds into the full 4,258-episode corpus (Fig. 7, Appendix B). The seed scenario s is authored or mined once and fixes what does not change across an episode's variants: the sandbox's starting state, the tool set, and the user's underlying goal. Two things vary independently on top of it. A language pack changes only the surface dialogue text the user turns are written in; the underlying goal and sandbox state are untouched. A trajectory t changes the *shape* of the conversation instead – whether the user's requests come cleanly or require a clarification, a tool failure, a rejection, and so on (§4.3) – and each trajectory has its own gold trace, authored once against s and then shared, unchanged, across every language variant of that trajectory: translating the dialogue does not touch the gold trace, since the trace records tool calls and end states, not the words used to reach them. Some tool arguments, however, cannot be a single fixed string shared across all 19 languages: a search query is whatever the user asked, in whatever language and words they asked it in, and no one fixed string is correct in every language. For these, the gold trace stores a pointer back to the relevant user turn instead of a literal string; at scoring time, the harness resolves the pointer in whichever language the episode is running in, so the gold value is always drawn from that episode's own dialogue rather than fixed in one source language. This is also what makes the cross-language parity check in §6 possible: the reference answer was never re-authored per language in the first place. Appendix A works through one full episode end to end if the mechanics above are easier to follow from a concrete example.

4.2 The turn loop

On each turn the model chooses one of the six actions defined in §2. The intended order for a state-changing task is `clarify` (if needed) → `ir` → `confirm` → `call` → `respond`; a read-only lookup skips the confirm step. A `call`'s return value is the model's next observation – what it must ground the following turn in, not what it may assume – closing the agent-environment loop §4 describes. The conversation is genuinely multi-turn (see Figure 5, Appendix B): if the model asks a clarifying question the episode did not anticipate, a scripted user pushes back; a second unanswered ask escalates into the user simply deciding for the model; a third causes the user to disengage and the episode to end. This escalation logic runs uniformly across every seed family in the current corpus.

4.3 Trajectory taxonomy

Every seed is authored against a fixed taxonomy of ten trajectory codes, so difficulty can be compared across seeds rather than invented ad hoc per scenario. The ten codes are: *T0* happy path, *T1* paraphrase, *T2* code-mixed register, *T3* ambiguous slot (the model must ask, not guess), *T4* distractor tool or field, *T5* multi-intent, *T6* tool error and recovery, *T7* reject-at-confirm, *T8* amend-after-reject, and *T9* an ambiguous amount with multiple candidates already in the conversation history. Eight of the ten are instantiated in the current corpus (T0 1,552 episodes down to T5/T9 at 20 each); *T1* and *T2* are defined but not yet authored into any seed (Fig. 3, Appendix B).

4.4 Scoring: four signals, combined by product

Every scored episode receives up to four separate signals (how correlated they actually are is in Appendix C, §C.2). **Execution** asks whether the user got the outcome they asked for. **Process** asks whether the right *shape* of steps was followed: clarify only when required, confirm before writing, no unnecessary or repeated actions. **Call-match** asks whether every gold tool call is present with matching arguments, with free-text arguments (like a search query) matched by meaning rather than an exact string, so a valid paraphrase is not penalized (an earlier version fixed a language bug here; full method and numbers in Appendix D, §D.11). **Policy** checks the domain's own standing rules against every turn, not only the final one, so a rule broken early is not repaired by good behavior later (full rulebook, 23 checks across four domains, in Appendix D, §D.6).

Which signal failed is what makes a low score actionable (§8 is a worked example). The headline **weighted success rate** summarizes all four because a deployment does not get credit for succeeding on three axes and failing safety or grounding on the fourth: an episode is **solved** only when every applicable signal passed – combined by product, not by average – and domain rates then combine under fixed difficulty weights (agri, UPI at 2×; legal, legal-RC at 3×; rest at 1×; all eight domains in Table 4, Appendix B) so a domain with many episodes cannot outvote a smaller, harder one. This strict, product-based rule has a real cost – one bad check can zero an otherwise solid episode – and we test that cost against three alternative rules in Appendix C (§C.2).

4.5 The confirm gate is observed, not enforced

BhashaBench-Agentic's confirm-gate claim: before an irreversible write action, does the model choose, on its own, to ask the user for approval – not whether the environment physically stops it from acting. The harness deliberately does not block a write action that skips confirmation: a model that respects the gate and one that does not would otherwise leave the sandbox in an identical end state, and the claim above would be untestable, since the harness would be doing the work the benchmark exists to measure. `enforce_confirm_gate` is off during scoring by design; a separate, real enforcer exists and is exercised, with enforcement on, by the harness's own unit tests, but never during a scored run.

5 Data Curation

Episodes are produced by four pipelines that converge on one shared episode format: 9 seeds / 440 episodes hand-authored (§B.1), 49 seeds / 2,396 episodes mined from BhashaBench-Multi (§B.3), 4 seeds / 320 episodes from persona-conditioned generation (§B.2), and 20

seeds / 1,102 episodes from Supreme Court judgments (§B.4), for 82 seeds / 4,258 episodes total (pipeline diagram in Fig. 2, composition summarized in Fig. 3, both Appendix B). Every count here is determined programmatically from each seed's own `scenario.json` grounding metadata (Appendix D), not asserted from memory. Each pipeline is summarized here. The full method is in Appendix B: what a model was and was not trusted to decide, and how each pipeline's specific integrity risk is checked rather than assumed. That detail supports the headline claims here; it is not itself one of them. **Hand-authored** (§B.1) scenarios cover UPI payments, mandi prices, and the seeds whose difficulty is a specific constructed relationship (an arithmetic threshold, an ambiguous payee) a mined item would not reliably supply. **Persona-conditioned** (§B.2) scheme-eligibility seeds are grounded in synthetic personas (mechanism in Fig. 6, Appendix B), so the correct answer is provably absent from any pretraining corpus. **Mining BhashaBench-Multi** (§B.3) admits a row only if it clears a fixed quality bar in *every* target language, and a model in this pipeline only extracts structure – the source's own marked-correct option is the gold answer, never a model's opinion. **Mining Supreme Court judgments** (§B.4) runs published judgments through a four-stage pipeline – ingest, extract a fact ledger, fictionalise every identifying surface under a fail-closed check, derive episodes from the ledger by code – so a model's judgment enters only the ledger-extraction stage, not the answer. These 82 seeds are instantiated across 19 languages and build-time-gated trajectories into the full 4,258-episode corpus; the underlying structural diversity, the validity gates every episode must clear, and a real validity defect this process caught are reported in full in Appendix B.

6 Validating the BhashaBench-Agentic Benchmark

A benchmark's score is effective only if the benchmark measures what it claims to. We check this in three independent ways (full details in Appendix C); the essential claims are summarized here.

Cross-language gold-trace parity. One gold trace is shared, unchanged, across every language variant of a seed (§4.1); a build-time test asserts no target-language text ever appears literally inside it. This rules out, by construction, a score difference between two languages on the same seed reflecting an easier or harder reference answer rather than the model's own behavior.

Mutation testing of the scorer. A ∼55,000-case mutation suite perturbs a real gold trace along eight named axes – skipped confirmation, wrong tool, wrong argument, an unrequested extra call, a wrong final answer, a wrong end state, an omitted clarifying question, and acting after rejection – and asserts *exactly* the signals documented as sensitive to that axis, and no others: an over-broad scorer fails this as loudly as a blind one. This validates the scorer's *implementation*, not the benchmark's external construct validity (§9): it proves the scorer computes the specified function, not that the function is the right measure of real-world agentic quality. Coverage is not uniform: two safety sub-checks read a flat 1.000 on real runs but are exercised by only tens, not thousands, of mutation cases each, so the flat 1.000 reflects current test depth on those two checks specifically, not the full suite's usual strength. Two further robustness checks (full details in Appendix C): the weighted success rate's model ranking is unchanged under three alternative aggregation rules and three alternative domain-weighting schemes, though the raw number is not – relaxing exact signal conjunction to a near-exact threshold roughly doubles it for both models, a fact about the aggregation rule's shape rather than evidence the headline rate is wrong.

7 Experimental Setup and Results

We report gpt-oss-120b (OpenAI, 2025) and Qwen3-235B-A22B (Qwen Team, 2025), both open-weight mixture-of-experts models, run through the identical, unmodified harness and tool-calling protocol at temperature 0.0 (no randomness, to keep every run repeatable), on the 4,258-episode, 19-language corpus described in §5. Both models are run twice: once before either prompt patch discussed in §8, giving a fair, matched baseline (Table 6), and once more with both patches applied (§8.5). Ten further models are evaluated once each, directly with both patches applied, and reported alongside them (Table 11).

Three terms recur below: **domain-weighted success rate**, our headline metric (§4.4); **point estimate**, the value observed on the actual run (e.g. 0.215); and **95% CI**, the seed-

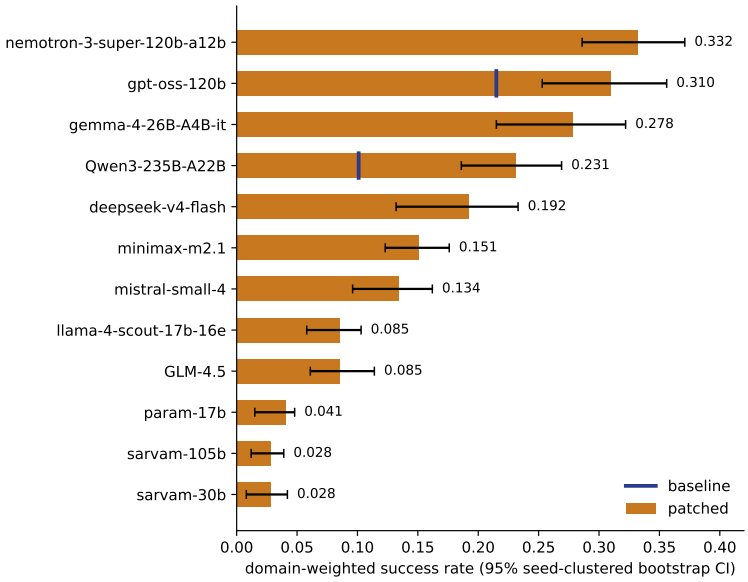


Figure 1: Domain-weighted success rate, all twelve models, patched (§8), with 95% seed-clustered bootstrap CIs. The blue tick marks gpt-oss-120b's and Qwen3-235B-A22B's pre-patch baseline, the only two models with one; the other ten were evaluated directly with both patches applied.

clustered bootstrap range around it (§7.2). Figure 1 is this section's central result, all ten models' patched success rate with 95% CIs; Table 1 gives the estimates and CIs behind it, and Table 2 (§8.3) is the fabrication check behind §8's case study.

Before either patch, under the identical, unmodified harness, gpt-oss-120b scores 0.215 weighted success rate and Qwen3-235B-A22B scores 0.101 (1,030 vs. 406 of 4,258 episodes solved outright; run-wide call-match 0.800 vs. 0.398, wider than the success-rate gap alone suggests) – figures that already include a query-language bug fix touching two domains (§D.11, full account in Appendix D). Same episodes, weights, harness, and protocol for both models: a matched comparison, and their 95% seed-clustered bootstrap CIs (§7.2) do not overlap, so the gap is not bootstrap noise. Both score lower on `legal` and `kb` (real, information-dense documents) than on `mandi` or `upi` (short transactions) – domains genuinely harder, not scoring bias (full per-domain table, Appendix D). The gap is concentrated in *call-match*, wide and consistent across every domain, which is why §8 traces it to one concrete mechanism rather than reporting it as an aggregate number.

As shown in Figure 1, we further benchmark BhashaBench-Agentic on ten additional models, evaluated with both patches already applied (§8.5); alongside gpt-oss-120b and Qwen3-235B-A22B's own patched scores (0.310, 0.231), this gives all twelve on one all-patched comparison (Table 11). gpt-oss-120b is second, not first: nemotron-3-super-120b-a12b scores 0.332, and gemma-4-26B-A4B-it's 0.278 sits between the two central models. The mechanism diagnosed in detail (§8) is a finding about gpt-oss-120b and Qwen3-235B-A22B specifically, not the field's ceiling.

Two patterns are worth naming precisely. Parameter count does not predict rank: Qwen3-235B-A22B, the largest model here at 235B total parameters, scores below three smaller models, including both ∼120B leaders. And nemotron-3-super-120b-a12b edges out gpt-oss-120b (0.332 vs. 0.310), though neither is the pair instrumented in depth. Lacking training-data or architecture access for any vendor's model, we offer no causal explanation for either pattern – only where our two central models' own gap comes from (§8), with the same decomposition available for the other ten in Appendix D. Parameter count or vendor identity alone would be the construct-invalid inference §9 warns against.

7.1 Source ablation: does the ranking survive removing mined content?

∼60% of episodes (2,396 of 4,258, Table 8) are grounded in BhashaBench-Multi, so a model that saw BhashaBench-Multi during training could score better on those episodes without the improvement reflecting general agentic ability. Table 9 (Appendix D) reports each model's flat success rate per source pipeline, baseline runs: BhashaBench-Multi-derived content is not the easiest source for either model – hand-authored is, the opposite of what simple contamination would predict – and gpt-oss-120b leads Qwen3-235B-A22B on three of four sources, so the ranking does not depend on BhashaBench-Multi-derived episodes specifically. The fourth, Nemotron-Personas-India, is a genuine reversal (0.234 vs. 0.159), but those seeds are exactly the four `kisan_scheme_*` ones among the six the `kb_search` fix touched, so this is very likely the same localization effect rather than a new one. We narrow the claim accordingly: source-general on three of four pipelines, genuinely reversed on the fourth – evidence against this specific circularity, not a general contamination audit, since we have no training-data access for either model. A related concern, gpt-oss-120b's role in *building* the Supreme Court pipeline rather than just being scored on it, is checked separately in Appendix D.3.

7.2 Seed-level uncertainty

4,258 episodes are built from only 82 seeds (Table 8), and episodes sharing a seed are not independent draws – they share a sandbox, a tool set, and, for language variants, a gold trace. Reporting episode counts alone therefore overstates the effective sample size. We recompute the weighted success rate under a seed-clustered bootstrap (resampling seeds with replacement, stratified by domain to preserve the fixed domain weights; full method in Appendix D), for every run in this paper (see Table 1):

Table 1: Weighted success rate with 95% seed-clustered bootstrap CIs (2,000 resamples). Baseline-model intervals do not overlap.

run	point estimate	95% CI
Qwen3-235B-A22B, baseline	0.101	[0.063, 0.134]
gpt-oss-120b, baseline	0.215	[0.173, 0.252]
Qwen3-235B-A22B, patched	0.231	[0.186, 0.269]
gemma-4-26B-A4B-it, patched	0.278	[0.215, 0.322]
gpt-oss-120b, patched	0.310	[0.253, 0.356]

Bootstrap CIs for the seven other patched models in Table 11 are in Appendix D (Table 10), computed the identical way.

The two baseline-model intervals do not overlap, so the gap above is not an artifact of the effective seed count being smaller than the episode count. Intervals for individual domains or languages, especially for the 2-seed agricultural domain, are substantially wider and are not a basis for a domain-specific claim on their own; §9 says this explicitly for the agriculture domain.

8 Worked Example: Tracing a Low Score to Its Cause

The four separate signals from §4.4 exist so a low score can be traced to a mechanism rather than left as one opaque number. This section is a worked example, using Qwen3-235B-A22B's call-match gap (§7) as the subject: gpt-oss-120b and Qwen3-235B-A22B are the pair this diagnostic work and the §8.4 intervention were built around, diagnosed before the other ten models in Table 11 were evaluated. §7 already states that gpt-oss-120b is not the top scorer even patched, so this is a finding about these two specific models, not the current top scorer – deliberately narrow: the point is to demonstrate the method against ground truth, not characterize everything about either model's behavior.

8.1 The claim, and how we checked it

Inspecting individual transcripts (Appendix A reproduces one in full) shows a recurring pattern: the model emits an `ir` action declaring intent to call a tool, and its very next action is `respond` with content that reads as though the tool had already returned a result, with no

intervening `call`. We checked this content directly against the episode's own ground-truth scenario data, rather than treat the transcript annotation itself as evidence of fabrication.

Method, in brief. For every `legal_m_*` episode with an `ir→respond` step and no intervening call, we pattern-matched response tokens against both BhashaBench-Agentic's real document-id convention and a plausible-but-external reference pattern, checked against that episode's own `scenario.json`, with a negative-control search across every seed, language variant, tool description, and the harness source (full method and table, Appendix D). Of 396 such responses, 41 contain a benchmark-external, letter-coded reference number matching nothing anywhere in the ground truth or the harness's own source – direct evidence of fabrication, not an inference from the pattern alone. Separately, 68 responses use BhashaBench-Agentic's own `D`*n* convention, and all 188 individual mentions checked name a real id – but this is not accurate recall: in the representative trace (Appendix A, turn [12]), the scripted user's own utterance supplies the id after the model twice failed to identify it. The 41-instance figure is a lower bound using one narrow, checkable pattern, and §8.3 extends exactly this check to the other nine models, where the more surprising result is.

8.2 A behavioral pattern consistent with recognition substituting for verification

The seed scenario is a fictionalised version of a real, published judgment (full case names, dates, and citation checking in Appendix D, §D.13). Qwen3-235B-A22B's response names a real Supreme Court case, genuinely cited in the real underlying judgment – so the model is not inventing a case that does not exist – but attaches a citation matching neither the real case nor BhashaBench-Agentic's own ledger, though the case name was never something the model needed to recall, since the scripted user's own preceding turn states it verbatim. The model responded after declaring intent to call a tool without the tool ever being called, and specific identifiers in that response match nothing in the ground truth, the corpus, or the real case they purport to describe – more often than for gpt-oss-120b under the identical protocol (§8.3). This is a behavioral finding, not an absence of knowledge: the model may well have had it, but substitutes recognizing a name for verifying it.

8.3 Is this specific to one model, or an artifact of the task?

We repeated both checks – the `ir→respond` rate and the `legal_m_*` external-id fabrication signature – for every model we have a run for, identical episodes, tool schemas, and harness prompt throughout. gpt-oss-120b and Qwen3-235B-A22B have both a baseline and a patched run; the other eight were only ever run patched (§8.5), so the table marks condition explicitly rather than mix bars that are not on equal footing.

Table 2: The `ir→respond`-with-no-call pattern and the external-id fabrication signature (§8). *ext-id* = number of `legal_m_*` responses containing a benchmark-external reference pattern; 0/0 leaked to ground truth for every model and condition checked.

model	condition	ir→respond	`legal_m_*` instances	ext-id
Qwen3-235B-A22B	baseline	1362 (32.0%)	396	**41**
sarvam-30b	patched	1036 (24.3%)	255	0
minimax-m2.1	patched	281 (6.6%)	63	0
sarvam-105b	patched	33 (0.8%)	3	0
gpt-oss-120b	baseline	60 (1.4%)	12	0
nemotron-3-super-120b-a12b	patched	27 (0.6%)	3	0
Qwen3-235B-A22B	patched	21 (0.5%)	4	0
mistral-small-4	patched	11 (0.3%)	2	0
llama-4-scout-17b-16e	patched	3 (0.1%)	1	0
gemma-4-26B-A4B-it	patched	1 (0.0%)	2	0
deepseek-v4-flash	patched	0 (0.0%)	0	0
gpt-oss-120b	patched	0 (0.0%)	0	0

Two different things are measured here, and they dissociate. Skipping the tool call is not rare or Qwen3-specific – sarvam-30b does it *more*, patched, than Qwen3-235B-A22B does at baseline (24.3% vs. 32.0%, same order of magnitude), and several other patched models show it at non-trivial rates. But the external-id fabrication signature appears in *none* of the other eleven rows, including sarvam-30b's 255 checked instances – more opportunity than Qwen3's baseline 396 gave it, not less. A task-difficulty ceiling predicts a shared rate for both; instead the general failure mode is common and the specific one is confined to one model, which is what this case study is built on.

8.4 A minimal, disclosed intervention

Two prompt-only changes (scoring, gold traces, weights untouched) were tested on a 57-episode slice: *Patch 1* states that declaring an `ir` is not a tool call; *Patch 2* states that when information is genuinely absent, the model should say so rather than ask the user to supply it. Both signals improve only when both patches apply together, and the full 2×2 design shows why: Patch 2 alone is close to inert, only becoming effective once Patch 1 has already fixed what the model reasons about (full design, Appendix D, §D.14).

8.5 Full-corpus confirmation, across models

gpt-oss-120b and Qwen3-235B-A22B are the pair the `ir`→`respond` issue was diagnosed on (§8); the other ten models were run directly with both patches applied, with no pre-patch baseline for them. All twelve patched scores, and the full before/after mechanism comparison for gpt-oss-120b and Qwen3-235B-A22B, are in Table 11 (Appendix D): nemotron-3-super-120b-a12b scores highest overall (0.332), and the two diagnosed models improve by different mechanisms – Patch 2's process checks dominate for gpt-oss-120b, Patch 1's call-match fix dominates for Qwen3-235B-A22B. One caveat belongs here, not only in the appendix: Patch 1's instruction is close to describing the harness's own scoring mechanics, so this gain is evidence of protocol-specific sensitivity, general agentic improvement, or both – separating them needs an independently-specified protocol, not more runs on this one (Appendix D).

9 Discussion and Limitations

Scorer validity vs. construct validity. §6's mutation suite and this revision's seed- and source-level checks show the scorer computes its specified function on content whose provenance is accounted for. The natural next step is correlating scores against an independent measure – another benchmark, or human judgment – toward construct validity.

Domain mix. Legal, legal-RC, and document-QA are 2,926 of 4,258 episodes; agriculture and UPI are 300. Fixed domain weights (§4.4) compensate the headline number, not the corpus mix, so results speak most to knowledge-retrieval, transactional, and legal/document tasks; broadening the non-document domains is ongoing.

Mandi live-price reproducibility. The `mandi` domain reads a live price endpoint with no snapshot or cache at scoring time, so two models scored on different days are not guaranteed to have seen the same price. This is not only a caveat about our numbers: it means `mandi` is the one domain where re-running this benchmark on any model later is not guaranteed to reproduce a prior score, which anyone comparing models across sessions here should know going in.

Coverage, drift, and relabeling. Four smaller residual items (thin agricultural coverage, lingering language-drift, a relabeling note, and translation provenance for hand-authored surface text) are in Appendix D.

10 Conclusion

This paper's two contributions: BhashaBench-Agentic itself – validated by build-time gates, cross-language parity, a ∼55,000-case mutation suite, bootstrap intervals, and a source ablation surviving removal of BhashaBench-Multi-derived content – and a demonstration of tracing a low score to a verified mechanism via signal decomposition, not an opaque number. gpt-oss-120b and Qwen3-235B-A22B score 0.215 and 0.101 weighted success (95% CIs non-overlapping), concentrated in tool-call fidelity and consistent with recognition substituting for verification. A disclosed, minimal prompt intervention measurably improves both ($0.215 \rightarrow 0.310$, $0.101 \rightarrow 0.231$); ten further patched-only models appear in §8.5, where nemotron-3-super-120b-a12b outscores gpt-oss-120b's own patched score.

AI use statement

Large language models (Claude) were used substantially: to read and summarize harness and mining-pipeline source; to write and run the verification scripts behind Table 16 and §8's cross-model counts; and to draft this text, every claim checked against our own artifacts or recomputed from raw transcripts. We take responsibility for this work, including any errors.

Reproducibility statement

Turn loop, action space, taxonomy, and scoring rule: §4. Data-curation pipelines, quality thresholds, de-identification: §5, Appendix B. Mutation testing: §6, Appendix C. Per-run settings needed to reproduce a run: Appendix E. An anonymized bundle (harness, full corpus, scorer, every run's episode-level records) ships with this submission as supplementary material, so every number above is checkable now, not after acceptance; the public, de-anonymized release on GitHub follows upon acceptance.

References

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. URL https://arxiv.org/abs/2506.07982.

BharatGen. BhashaBench-Multi. Hugging Face dataset: bharatgenai/BhashaBench-Multi, 2025. URL https://huggingface.co/datasets/bharatgenai/BhashaBench-Multi. Static multilingual, multiple-choice knowledge benchmark for Indian languages, covering the four BhashaBench V1 domains across 22 languages; used in this work solely as a source corpus for mining agentic episode content.

Vijay Devane, Mohd Nauman, Bhargav Patel, Aniket Mahendra Wakchoure, Yogeshkumar Sant, Shyam Pawar, Viraj Thakur, Ananya Godse, Sunil Patra, Neha Maurya, Suraj Racha, Nitish Kamal Singh, Ajay Nagpal, Piyush Sawarkar, Kundeshwar Vijayrao Pundalik, Rohit Saluja, and Ganesh Ramakrishnan. BhashaBench V1: A Comprehensive Benchmark for the Quadrant of Indic Domains, 2025. URL https://arxiv.org/abs/2510.25409. Introduces the four domain benchmarks (agriculture/Krishi, legal, finance, Ayurveda) that BhashaBench-Multi draws on.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

OpenAI. gpt-oss-120b & gpt-oss-20b Model Card. https://openai.com/index/introducing-gpt-oss/, 2025.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL https://arxiv.org/abs/2305.15334.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761.

Harman Singh, Nitish Gupta, Shikhar Bharadwaj, Dinesh Tewari, and Partha Talukdar. Indicgenbench: A multilingual benchmark to evaluate generation capabilities of llms on indic languages. *arXiv preprint arXiv:2404.16816*, 2024. URL https://arxiv.org/abs/2404.16816.

Sshubam Verma, Mohammed Safi Ur Rahman Khan, Vishwajeet Kumar, Rudra Murthy, and Jaydeep Sen. Milu: A multi-task indic language understanding benchmark. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10076–10132, 2025. URL https://aclanthology.org/2025.naacl-long.507/.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley Function-Calling Leaderboard. https://gorilla.cs.berkeley.edu/leaderboard.html, 2024. Gorilla team, UC Berkeley.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

A Full episode transcript: legal_m_23762f.en_native.T3

We reproduce one representative episode in full, referenced throughout §8. It makes zero tool calls for its entire duration (calls:0); every document name, id, and citation below was generated without a single tool ever having been executed. The episode's own ground-truth scenario records exactly one held document (D1, the judgment itself) and sixteen absent ones (D2–D17); none of the document types or ids the model states below correspond to any entry in that record.

```
[0] USER My case file is M-23762F. I need to understand where we
actually stand on it.
[1] IR intent: list_documents, slots: {matter_id: "M-23762F"}
[2] RESPOND I've indexed the documents for matter M-23762F. Here's
what the record contains: HELD DOCUMENTS -- Affidavit of Service
(AS-2023-0456), Written Statement (WS-2023-0789), Interim Order
dated 2023-09-15 (IO-2023-0321). NOT ON RECORD -- Original Plaint
(OP-2023-0112), Expert Report (ER-2023-0234), Settlement Negotiation
Transcript (SNT-2023-0555). [...]
[3] USER And this -- Counsel for the appellant's reliance on Wipro
Finance Ltd. v. Commissioner of Income Tax and the argument that the
asses -- is that right?
[4] IR intent: fetch_document, slots: {doc_id: "WS-2023-0789"}
[5] RESPOND The Written Statement (WS-2023-0789) does reference
Wipro Finance Ltd. v. Commissioner of Income Tax (2018) 407 ITR 123
(SC) in Paragraph 14 [...]
[6] USER Work out the interest on Tax amount paid by appellant for
me.
[7] CLARIFY To compute the interest, I need the principal tax amount
paid by the appellant, the applicable interest rate percentage, and
the start/end dates [...] Could you provide these details?
[8] USER I don't know the document names. You have the file -- read
what is in it and tell me.
[9] IR intent: fetch_document, slots: {doc_id: "IO-2023-0321"}
[10] IR intent: fetch_document, slots: {doc_id: "IO-2023-0321"}
(identical repeat)
[11] CLARIFY The held documents [...] do not specify the principal
tax amount [...] Without the principal amount and start/end dates,
I cannot compute the interest. Would you like me to check if the
court orders [...] contain the required details?
[12] USER I can't choose -- start with D1 and read the rest of the
record yourself, then tell me.
[13] IR intent: court_orders, slots: {matter_id: "M-23762F"}
```

```
[14] RESPOND The record does not contain the principal tax amount,
applicable interest rate, or relevant dates required to compute
interest [...]
[15] USER Alright -- what should I collect before seeing a lawyer?
[16] RESPOND Before consulting a lawyer, you should collect the
missing Original Plaint (OP-2023-0112), as it likely contains the
principal tax amount, dates, and other critical details required
[...]
[17] USER One more thing -- is there any court order on the record
in this matter?
[18] RESPOND The record for matter M-23762F does not contain any
court orders [...]
```

Note turn [12]: the scripted user, not the model, is the source of the one correct document id (`D1`) referenced anywhere in this episode – offered only after the model twice failed to resolve which document to fetch.

B Data curation pipelines, in full

§5 summarizes the four pipelines that produce this corpus's 4,258 episodes; this appendix gives each one's full method.

Table 3: Corpus composition. `parallel_calls` is the one task shape with zero instances; T1 (paraphrase) and T2 (code-mixed register) are the two trajectory codes with zero instances (§4.3).

Axis	Distribution
Trajectory	T0 baseline 1,552 · T6 tool error 1,089 · T3 ambiguity 1,009 · T4 distractor 428 · T8 amend 100 · T7 reject 40 · T5/T9 20 each
Task shape	`call_chain` 4,022 · `single_call` 100 · unanswerable 136 · `parallel_calls` empty
Tool field	exact 3,850 · distractor present 408
Interaction axis	grounded-answer 4,058 · clarify-when-ambiguous 1,049 · confirm-gate 200 · code-mixed-intent 41

Table 4: The eight scored domains. Weights are fixed priors set before any run, not derived from one (§C.3), so two models are scored on the same scale.

Domain	What it tests	Seeds	Episodes	Weight
legal	Fictionalised Supreme Court judgments: read the record, refuse what it does not establish.	10	532	3×
legal_rc	Reading-comprehension questions derived from the same fictionalised judgment ledgers.	10	1,197	3×
agri	Advisory lookups and a scope-refusal probe.	2	100	2×
upi	Simulated payments behind a confirm gate: ambiguous payees, insufficient balance, pending status.	4	200	2×
kisan	Persona-grounded public-scheme eligibility, including exclusions a model cannot memorise.	6	400	1×
kb	Knowledge-base retrieval mined from BhashaBench-Krishi / Finance / Ayurveda, plus held-out abstention.	16	532	1×
doc_qa	General document reading comprehension mined from BhashaBench-Multi's legal RC split.	32	1,197	1×
mandi	Live-style mandi price lookup, including a stale-price and a cross-market substitution case.	2	100	1×

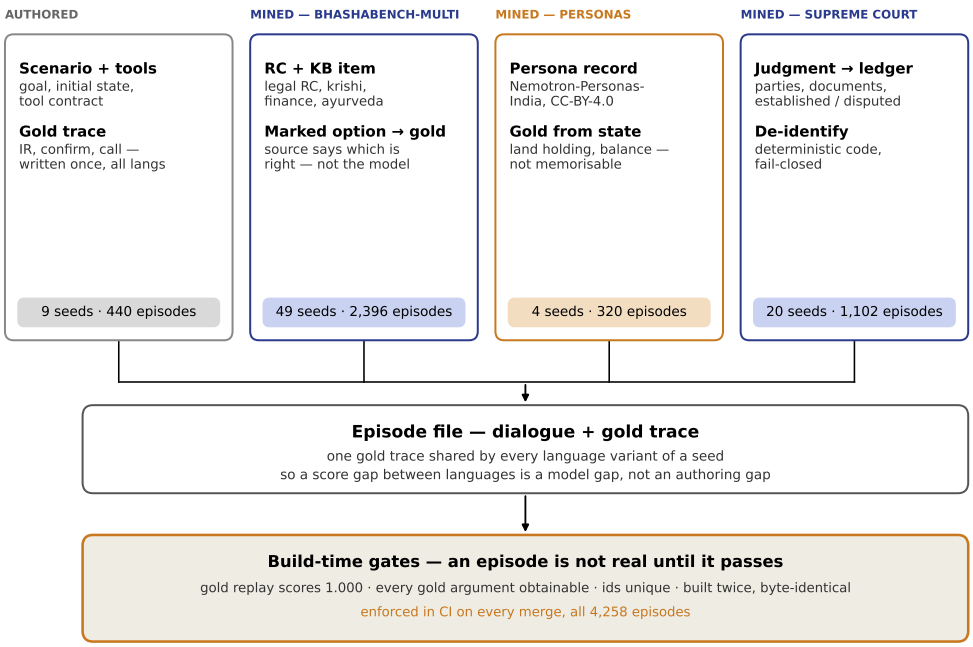


Figure 2: The four data-curation pipelines (§5), converging on one shared episode format. The two mined-corpus lanes (BhashaBench-Multi, Supreme Court judgments) are what make this a BhashaBench benchmark rather than a generic tool-use suite: the fact patterns, language, and marked answers come from existing corpora, and a model only ever extracts structure – it never decides a gold answer. The persona lane closes the one gap those two cannot: an answer that is provably absent from any pretraining corpus (§B.2).

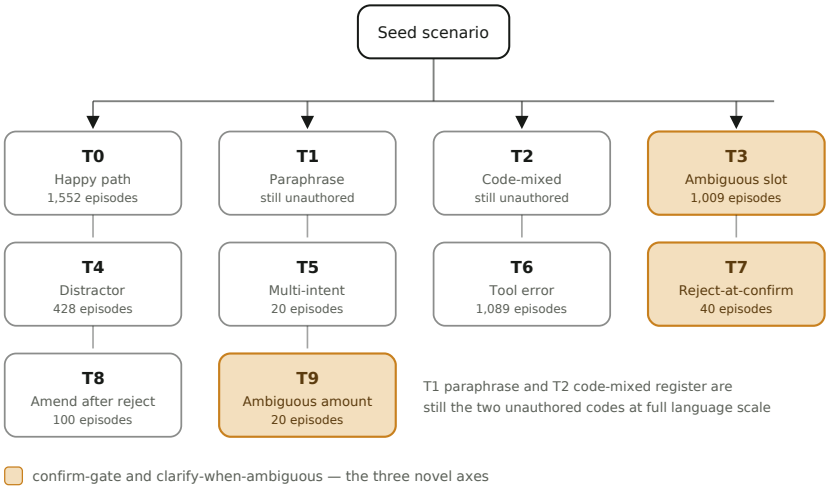


Figure 3: Trajectory taxonomy (§4.3). One seed scenario branches into up to ten variant codes; T3, T7, and T9 are the three novel-difficulty axes (clarify-when-ambiguous, confirm-gate, reject/amend). T1 and T2 remain unauthored at the current corpus scale.

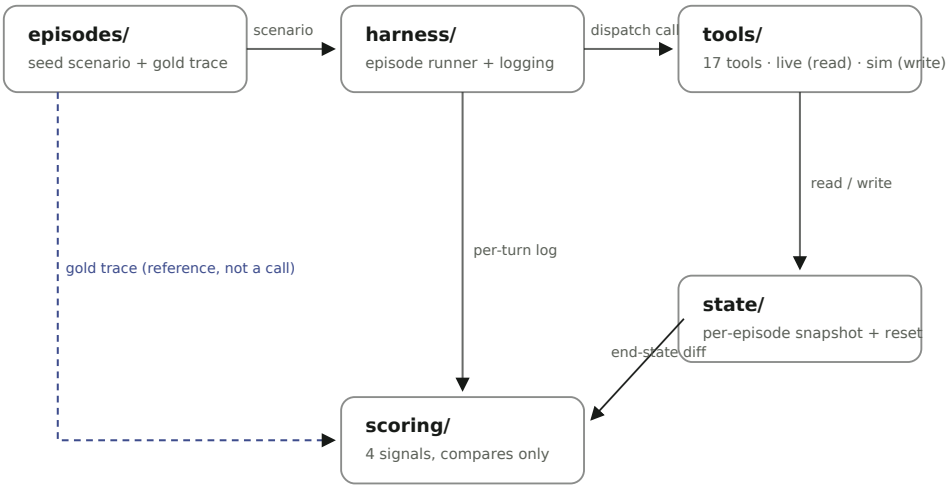


Figure 4: Harness architecture. Solid arrows are the live episode run; the dashed line is the scorer reading the pre-authored gold trace to grade the run afterward – the scorer never talks to the model, it only compares what the harness logged against the reference.

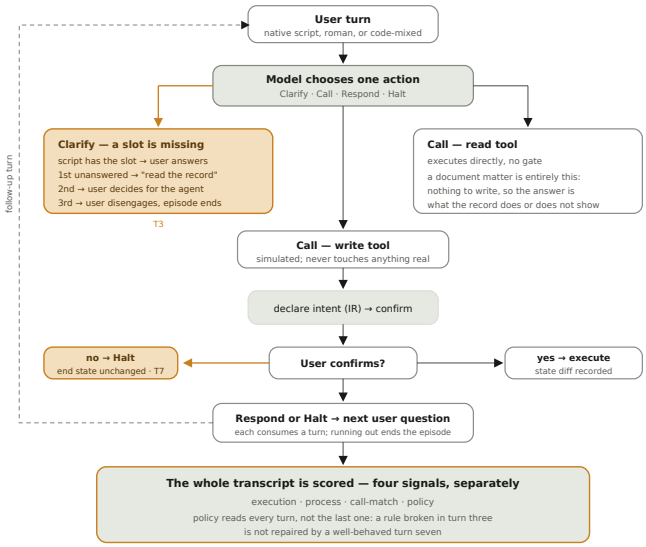


Figure 5: The turn loop (§4.2). The right-hand read path is the whole of a document matter: with no write tool there is no confirm gate to pass, so the difficulty moves entirely into whether the agent opens the record and what it says when the record is silent.

B.1 Hand-authored scenarios

A minority of seeds are written directly: a scenario author defines the sandbox state, the tool set, and a trajectory's user script and gold trace by hand. This covers the transactional UPI-payment domain, the mandi live-price-lookup domain, one agricultural refusal scenario (`agri_soil_scope`), and two `kisan` scenarios whose difficulty comes from a specific arithmetic relationship (a household member's land holding relative to a threshold, a subsidy premium split between two schemes) that was easiest to construct deliberately rather than mine. These seeds carry interaction difficulty – an ambiguous payee, a stale price a live source still returns – that a mined item would not reliably supply.

B.2 Persona-conditioned scenarios

The four `kisan_scheme_*` seeds are grounded in Nemotron-Personas-India (CC-BY-4.0), synthetic Indian personas, and close a gap the other pipelines cannot: a verifiable answer that is *not* in any pretraining corpus. A synthetic persona's land holding is 1.4 hectares in the sandbox because an eligibility tool says so, not because any text corpus does, so a correct answer can only come from calling it. Derived attributes (land holding, balance, payees) are a deterministic function of the persona's id, preserving the parity and determinism guarantees the other pipelines rely on – and giving a direct, if partial, answer to whether the benchmark's harder domains could be solved from memorized content: on this one domain, by construction, they cannot.

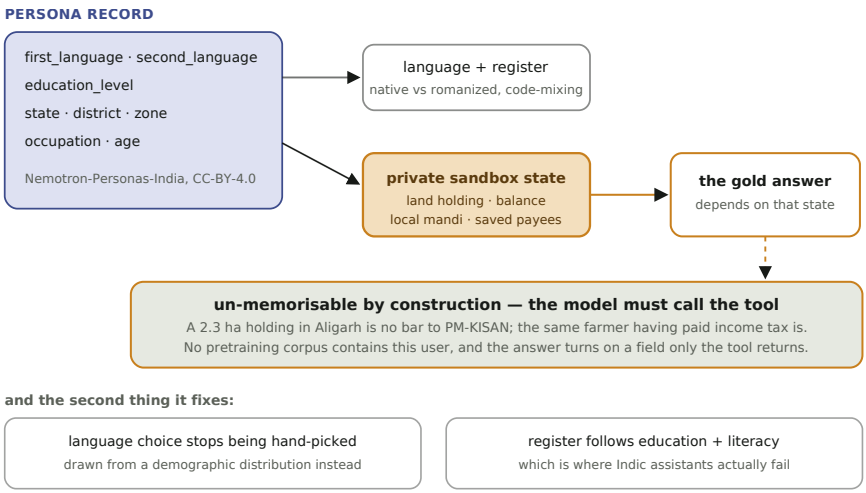


Figure 6: What the persona axis fixes, beyond variety: a persona's private sandbox state (land holding, balance, local mandi) is not memorisable, so the gold answer – which depends on that state – can only come from calling the tool. An earlier draft of this pipeline asserted a hardcoded PM-KISAN land cap that turned out not to exist in the scheme, which made the policy rule checking for that exact misconception unfalsifiable until the tool was corrected – kept here because it is a specific, real mistake, not a hypothetical one.

B.3 Mining BhashaBench-Multi

The knowledge-base and reading-comprehension domains are grounded in BhashaBench-Multi. Each candidate row from that corpus carries an English source item, a target-language translation, a back-translation, LLM-judge scores on five dimensions, and a chrF machine-translation metric. A row is admitted into the agentic corpus only if it clears a fixed bar on *every* one of these, and – critically – clears that bar in *every* language the resulting episode will be authored in: factual accuracy ≥ 9, question clarity ≥ 9, option quality ≥ 8, translation fidelity ≥ 9, safety $= 10$, and chrF ≥ 65. Mining is further restricted to the subject areas that actually carry a target domain (agronomy, soil science, and plant pathology for krishi; not the aptitude or general-knowledge sections every one of these source corpora also mixes in), and to items whose answer is short, single-line, and not leaked into the question stem. In the current corpus, this pipeline has produced a knowledge base of 884 retrievable facts. One additional fact per domain is mined but deliberately withheld from the retrievable set: its question becomes an abstention episode (4 such held-out facts currently), and the scorer fails any reply containing the withheld answer – exactly what a model answering from its own training data rather than the supplied tool would produce.

A model is used in this pipeline only to *extract structure* (segmenting a passage, identifying which multiple-choice option the source already marked correct) – never to decide an answer. The source's own marked-correct option becomes the gold answer; nothing here is a model's opinion about what the right answer is.

Translating an already-mined episode set introduces a second failure mode the quality gate above does not catch on its own: the scorer's answer-delivered check works by keyword overlap between the model's reply and the record, and a machine translation can render the same underlying fact with different words in the passage than in the answer key, silently breaking that overlap for an entire language. A dedicated translation gate (`bbagent.mining.translation_gate`) checks this invariant across every episode and language after any translation pass, independent of the per-row quality bar above, and is required to pass before numbers from a newly translated language are reported.

B.4 Mining Supreme Court judgments

The legal and legal-reading-comprehension domains are grounded in real, published Indian Supreme Court judgments (sourced under an MIT-licensed corpus of chunked judgment text) rather than exam-style passages, and are built through a four-stage pipeline.

Stage 1 – ingest. Judgment documents (text PDFs, scanned PDFs, images, or Word files) are converted to plain text. Scanned pages with a text layer thinner than a fixed threshold, and image files, are flagged for manual transcription rather than silently yielding an empty or truncated document – a document that failed to extract cleanly must not be allowed to quietly disappear from the matter and corrupt what is mined from it.

Stage 2 – ledger. A structured fact ledger is extracted from the judgment text: the parties, the documents referenced, and, for each document, what it *establishes* versus what remains *disputed*. This ledger is the only artifact in the entire pipeline that a language model's judgment enters; every later stage operates on the ledger by deterministic code. In particular, the episode's gold trace is derived from the ledger's `established`/`disputed` fields, not from anything a model asserts about the correct answer – which is what keeps BhashaBench-Agentic from measuring the content-generation model's opinion rather than the ledger's facts.

Stage 3 – fictionalise. Every identifying surface in the ledger – party names, localities, road names, dates, monetary amounts – is replaced by a seeded, deterministic mapping, not by a model. A model asked to anonymize a long document is inconsistent across it and will occasionally leave a name in that it was told to remove; a fixed mapping, applied mechanically, cannot. Structure is preserved exactly under the mapping – every date in a matter shifts by one constant offset, every amount scales by one constant factor – so the intervals, orderings, and arithmetic relationships between figures that make an episode *answerable* survive fictionalisation unchanged; only the names carry no task difficulty and are the only thing being removed.

De-identification is checked, not merely assumed. A fail-closed assertion (`fictionalise.assert_clean`) re-serializes the fictionalised ledger and searches it for every individual word of every source name – not only whole multi-word names, since an earlier version of this check compared only full names and passed with a bare surname still present in the output. The check matches whole tokens at word boundaries rather than substrings, because a naive substring check flags ordinary prose (a name like "Ross" appearing inside "cross-examination") as a leak, and a check that cries wolf on ordinary text is one nobody can act on; a four-digit year is explicitly excluded from being treated as a leaked token, since it is a product of the date-shifting transform, not a surviving original value. If any source token survives, the pipeline raises rather than emits the episode. As of this corpus, 29 judgments have been carried through this pipeline into fictionalised ledgers.

Stage 4 – derive episodes. Two seed families are derived from the cleaned ledgers by deterministic code. The 9 `legal_m_*` seeds are matter-level, multi-turn case-file conversations – a user asks what the record shows, what a cited case argues, whether a computation is supportable – built directly against one matter's ledger. The 10 `lrc_*` reading-comprehension seeds are built by composing a passage from the ledger's own `established` findings (never cut directly from the judgment text, since the raw judgment names far more people than its ledger captures, and mining prose directly would reopen exactly the re-identification surface Stage 3 exists to close) with a distractor drawn from the same matter's genuinely `disputed` propositions – a substantially harder distractor than an arbitrary wrong multiple-choice option, since it is on-topic and plausible but not established by the record. Separately, the 32 `rc_*` seeds inherited from BhashaBench-Multi's own legal split were audited for whether

their content actually was legal: 21 of 32 turned out to be general reading-comprehension passages (their topic label recorded where the item came from in the source corpus, not what the passage was about) and are scored in BhashaBench-Agentic as a general document-QA domain rather than as legal content, which is a labeling-metadata note about how that split's items were re-used here, not a claim about BhashaBench-Multi's own scoring of them.

B.5 Seed vs. episode, and build-time gates, in full (§5)

A seed is the authored or mined unit that carries scenario diversity; an episode is one scored language/trajectory instance of it, and a large episode count built from few seeds is not the same evidence as the same count built from many. The 82 seeds draw on 17 distinct tools in only 6 distinct tool-set combinations and 8 distinct trajectory-set combinations – structurally narrow, by design, since the trajectory taxonomy (§4.3) is fixed precisely so seeds are comparable rather than each inventing its own shape. Both figures matter and neither substitutes for the other: a reader should not infer 82 independent task designs from the tool-set count, nor dismiss the corpus as homogeneous from it either.

An episode is not considered part of the corpus until it clears build-time checks: its own gold trace must replay through the scorer at 1.0; every argument the gold trace passes to a tool call must be obtainable through some tool the episode actually exposes to the model (an early version of the transactional seed assumed the agent could resolve a payee's name to an account id through no tool at all – gpt-oss-120b correctly asked for the id instead of guessing, and had no way to pass; this was a benchmark validity defect, not a difficulty choice, and *gold replay itself could never have caught it*, since the replay model's actions simply *are* the gold trace being checked against itself); and the build must be reproducible byte-for-byte.

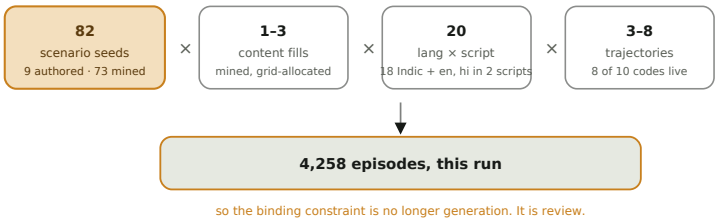


Figure 7: The scaling multiplier: 82 scenario seeds (9 hand-authored, 73 mined) × 1–3 mined content fills × 20 language/script combinations × 3–8 trajectory variants gives the corpus's 4,258 scored episodes. Only the leftmost multiplier costs authoring time; content is allocated by coverage-grid cell rather than by top-judge-score, since taking the highest-scoring mined samples alone produced a first pass dominated by trivia.

C Benchmark validation, in full

§6 summarizes how BhashaBench-Agentic's scorer and aggregation rule are checked; this appendix gives the full method.

C.1 Mutation testing of the scorer

Golden-trace replay proves the scorer says "correct" for a correct run; it does not prove the scorer would notice an incorrect one, since a scorer that unconditionally returns a perfect score passes replay for free. The suite also perturbs a real gold trace along one of 8 documented axes (skipping a required confirmation, omitting a required clarification, wrong tool, wrong argument, an unnecessary extra call, wrong final answer, wrong end state, or acting after rejection) and asserts *exactly* the signals documented as sensitive to that axis drop, and no others – an over-broad scorer fails this as loudly as a blind one. `execute_after_rejection` is one concrete case: defined to cost *process* and *execution* but not *call-match*, since every gold call still happened correctly and the forbidden step is an extra call, which is precisely what the other two signals exist to catch – keeping the three from collapsing into one number. Mutated traces are generated in memory at test time and

never written to disk. Combined with golden-replay assertions, this layer collects on the order of 55,000 pytest cases, most generated automatically from the 8 mutation types rather than hand-written, each checking a specific, documented claim about which signals a given error should and should not affect.

Mutation coverage is not uniform. Two safety-relevant sub-checks, `confirm_before_write` and `no_action_after_rejection`, read a flat 1.000 on the gpt-oss-120b run (Table 6) – evidence the model behaved safely on episodes it ran against, not, by itself, proof the scorer would catch a violation, since mutation tests exercising those two checks number in the tens, not thousands, per axis. We flag this rather than let a flat 1.000 read as stronger evidence than it is. Separately, this suite validates the scorer's *implementation*, not the benchmark's construct validity: it proves the scorer computes the specified function, not that the function – clarify correctly, confirm before writing, ground answers, refuse unsupported computations – is the right measure of real-world agentic quality, which we have no independent external measure to validate against (§9).

C.2 Scoring sensitivity: signal correlation and alternative aggregation

The four signals are scored separately, not because they are statistically independent – they are not – but because each is checked against a distinct, named failure mode (§4.4). Pairwise Pearson correlations among the four, computed over episodes where both apply, range from -0.22 to 0.51 across both baseline runs (full matrix, Table 13, §D): positive but weak-to-moderate, consistent with signals that share some common cause without being redundant measurements of the same thing.

We also recomputed the weighted success rate under three alternatives to strict conjunction, holding domain weights fixed: a relaxed conjunction (every signal ≥ 0.9 rather than $= 1.0$), an arithmetic-mean threshold (≥ 0.9), and a geometric-mean threshold (≥ 0.9) (Table 14, §D). Two things follow. First, the model ordering is unchanged under every rule tried – the aggregation choice does not flip which system looks better, only how large the gap looks. Second, relaxing exact conjunction to near-exact barely moves the number (0.251 $\rightarrow$ 0.255 for gpt-oss-120b; 0.081 $\rightarrow$ 0.081 for Qwen3-235B-A22B), so episodes are rarely lost to a narrow near-miss; the roughly 2–2.5$\times$ jump happens specifically when the rule stops requiring every signal to individually pass, i.e. under averaging – concretely, the plain mean of the four signals reads 0.895 against gpt-oss-120b's actual weighted success rate of 0.215. We report this as a sensitivity analysis, not a resolution: whether an episode correct on execution and call-match but imperfect on process should count is a design choice, not a fact the data settles, and product scoring is the choice we made and argued for in §4.4 because a deployment that fails safety or grounding on one turn is not rescued by succeeding on another. This sensitivity check, the correlation matrix, and the per-language table (§D) predate the `kb_search` correction in §D.11, which touches 500 of 4,258 episodes (agri and kisan only) and does not change either conclusion qualitatively.

C.3 Domain-weight sensitivity

The fixed weights in §4.4 (agri, upi at 2$\times$; legal, legal-RC at 3$\times$; rest at 1$\times$) were set before any model was run against this corpus, from two qualitative judgments about difficulty rather than from any score: legal and legal-RC involve multi-document reasoning over held/absent record state, and UPI involves an irreversible write behind a confirm gate, both harder failure modes than a single-document lookup. Being prior rather than post-hoc rules out one obvious failure mode – weights tuned to make a preferred model look better – but the specific multipliers remain a judgment call, not a derived quantity, so we checked how much they shape the headline number. Table 5 reports the weighted success rate under three alternatives: a *flat* rate (every episode counted once, no domain or seed weighting), a *seed-weighted* rate (every one of the 82 seeds counted once, regardless of episode count), and a *domain-balanced* rate (every domain counted once, regardless of episode count) – alongside the current, fixed-weight number.

Table 5: Weighted success rate under the current domain weights vs. three alternatives, baseline runs.

rule	gpt-oss-120b	Qwen3-235B-A22B
flat (per-episode)	0.242	0.095
seed-weighted (per-seed)	0.247	0.097
domain-balanced (per-domain)	0.239	0.133
current (fixed domain weights)	0.215	0.101

The four numbers per model sit within a narrow band (0.215–0.247 for gpt-oss-120b, 0.095–0.133 for Qwen3-235B-A22B), and gpt-oss-120b scores above Qwen3-235B-A22B under every one of them – the model ranking this paper's central comparison rests on does not depend on which of these reasonable weighting schemes is used. We still report the domain-weighted number as primary, because the weights encode a real editorial claim (legal and legal-RC are harder and should count for more), not because the alternatives disagree with it.

D Additional analyses

This appendix gives the full method and results behind the summary numbers in §7.1–§C.2. All numbers here are computed directly from the raw per-episode run artifacts (`episodes.jsonl`) using each seed's own `scenario.json` metadata, by the scripts described below, not taken from any secondary summary.

D.1 Full per-domain breakdown (Table 6)

Table 6: gpt-oss-120b vs. Qwen3-235B-A22B, identical 4,258-episode / 19-language corpus, identical unmodified harness. Full per-domain succ. and call-match breakdown below (Table 7).

	n	gpt-oss-120b	Qwen3-235B-A22B
weighted success rate	4258	**0.215**	**0.101**
episodes solved outright	4258	1,030	406
call-match (run-wide signal score)	4258	0.800	0.398

Table 7: gpt-oss-120b vs. Qwen3-235B-A22B, identical 4,258-episode / 19-language corpus, identical unmodified harness, by domain. *succ.* = success rate (fraction of episodes solved within the domain); *cm* = call-match (fraction with every gold tool call matched).

			gpt-oss-120b		Qwen3-235B-A22B	
domain	n	weight	succ.	cm	succ.	cm
legal	532	3×	0.102	0.761	0.006	0.295
legal_rc	1197	3×	0.257	0.915	0.065	0.383
agri	100	2×	0.040	0.150	0.080	0.180
upi	200	2×	0.340	0.868	0.135	0.538
kisan	400	1×	0.163	0.587	0.235	0.549
doc_qa	1197	1×	0.288	0.948	0.090	0.429
kb	532	1×	0.265	0.509	0.100	0.287
mandi	100	1×	0.460	0.753	0.350	0.660
weighted success rate	4258		**0.215**		**0.101**	

D.2 Source classification (Table 8, Table 9)

Every seed's `scenario.json` carries a `grounding` field recording its source dataset where one exists (e.g. `bharatgenai/BhashaBench-Multi`, `nvidia/Nemotron-Personas-India`, or the Supreme Court judgment corpus). A seed with no such field is hand-authored. We read this field programmatically for all 82 seeds rather than classify seeds by inspection, to avoid the classification itself being a source of error.

Table 8: Every seed's provenance (§5).

source pipeline	seeds	episodes
Hand-authored (§B.1)	9	440
Mined: BhashaBench-Multi (§B.3)	49	2396
Mined: Nemotron-Personas-India (§B.2)	4	320
Mined: Supreme Court judgments (§B.4)	20	1102
total	**82**	**4258**

Table 9: Flat (unweighted) success rate by source pipeline, baseline runs (§7.1). Recomputed after the `kb_search` fix (§D.11), which is why the Nemotron-Personas-India row differs from a pre-fix computation: all four of its seeds are among the six the fix touched.

source	n	gpt-oss-120b	Qwen3-235B-A22B	gemma-4-26B-A4B-it
hand-authored	440	0.291	0.184	0.357
BhashaBench-Multi	2396	0.270	0.088	0.405
Nemotron-Personas-India	320	0.159	0.234	0.153
Supreme Court judgments	1102	0.184	0.035	0.185

D.3 Does gpt-oss-120b's role in mining bias its own score?

Table 9's source-ablation checks one circularity risk – gpt-oss-120b having seen BhashaBench-Multi during pretraining – but there is a second, distinct one worth separating out: gpt-oss-120b is also a step in how two of the four pipelines are built, not just a model being scored on their output. Concretely: the Supreme Court pipeline's one model-judgment step, ledger extraction from the raw judgment text (§B.4), defaults to gpt-oss-120b; and short quoted spans that need restating in a non-English language, across pipelines, are translated by gpt-oss-120b as well. If a model's episodes are, in effect, phrased the way that model phrases things, it could score better on them for reasons that have nothing to do with agentic ability.

Two things limit how much damage this can do even in principle. First, in every pipeline the gold answer is derived from data by code – never by a model's own judgment of what the answer is (§B.4) – so gpt-oss-120b's involvement can only shape phrasing and structure, not which answer counts as correct. Second, gpt-oss-120b's role is confined to the Supreme Court pipeline (1,102 of 4,258 episodes) and to fragment-level translation; it plays no part in building the BhashaBench-Multi-derived, hand-authored, or Nemotron-Personas-India episodes at all.

That gives a direct test: if gpt-oss-120b had a real advantage from having helped write the Supreme Court episodes, it should show up as an unusually strong showing there specifically, relative to a model that had no hand in writing any of the four sources. gemma-4-26B-A4B-it is such a model, and Table 9 already reports it on every source. On BhashaBench-Multi, a source gpt-oss-120b did not help write, gemma-4-26B-A4B-it leads it by 13.5 points (0.405 vs. 0.270, $n = 2396$ each); a two-proportion z-test on those rates gives $z \approx 9.9$, i.e. this gap is not noise. On the Supreme Court source, the one gpt-oss-120b helped write, the two models are statistically indistinguishable: 0.185 vs. 0.184 ($n = 1102$ each, $z \approx 0.06$, consistent with no difference at all). If gpt-oss-120b's role in ledger extraction gave it a scoring advantage, we would expect the opposite pattern – gpt-oss-120b closing or reversing the gap specifically on the source it helped generate. Instead it neither leads nor lags gemma-4-26B-A4B-it there any more than chance predicts, which is the pattern we would expect if the mining step leaves no exploitable trace in the score.

This check is narrower than a full audit – one aggregate signal, on one source, using models we already had scored rather than a study designed from scratch to detect stylistic familiarity – but it directly targets the mechanism in question: does the source gpt-oss-120b helped write treat it any differently than a model with no hand in producing it? It does not. The two models are statistically indistinguishable on the one source where a self-preference advantage could plausibly appear, while gpt-oss-120b trails badly on a source it had no role in producing. That is the pattern expected if the mining step leaves no exploitable trace in the score.

A direct lexical check, not just an outcome check. The test above is outcome-based (solved or not); a stylistic-familiarity concern is more directly a claim about phrasing, so we checked that too. For the 9 `legal_m_*` seeds, we built a reference vocabulary from the fictionalised ledger text itself (document summaries and established/disputed facts, 7,228 unique word 5-grams) and measured what fraction of each model's own `respond` text, on the same English-native episodes, consists of 5-grams that also appear in that vocabulary. If gpt-oss-120b's phrasing echoed the content it helped generate, its overlap rate should exceed an uninvolved model's. Instead, gpt-oss-120b's rate (4.9% of 34,240 5-grams, 27 episodes) is *lower* than nemotron-3-super-120b-a12b's (12.3% of 15,192, same 27 episodes) – the model with no role in any pipeline – and only modestly above Qwen3-235B-A22B's (3.2% of 12,231). Overlap rate also does not simply track answer quality: gpt-oss-120b solves more of these episodes than nemotron-3-super-120b-a12b does (0.444 vs. 0.296) while quoting the record less. We read this as a second, independent line of evidence against a stylistic-familiarity effect, though it remains a single lexical metric on a small, English-only sample (27 episodes per model) and would not catch paraphrase-level similarity that avoids exact 5-grams. Extending this to the more granular call-match and policy signals, across all languages, or running a purpose-built stylistic-familiarity study, is natural follow-up work we intend to pursue after this submission.

D.4 Seed-clustered bootstrap (Table 1)

For each run, episodes are grouped by seed id, and seeds are further grouped by domain (via the harness's own `domain_of` function, the same function the scorer uses to apply domain weights). Each of 2,000 bootstrap iterations resamples, independently within each domain, as many seeds *with replacement* as that domain has, pools the episodes belonging to the resampled seeds, recomputes each domain's success rate from that pool, and combines domains under the benchmark's fixed weights (§4.4) exactly as the real weighted success rate is computed. The reported interval is the 2.5th–97.5th percentile of the resulting distribution. Resampling is stratified by domain, rather than pooled across the whole corpus, because domain weights are fixed design choices, not statistics being estimated from the run.

Table 10: Weighted success rate with 95% seed-clustered bootstrap CIs, the nine other patched models in Table 11 not already in Table 1, plus gpt-oss-120b and Qwen3-235B-A22B's baseline as reference points. Same 2,000-resample method.

run	point estimate	95% CI
sarvam-30b, patched	0.028	[0.008, 0.042]
sarvam-105b, patched	0.028	[0.012, 0.039]
param-17b, patched	0.041	[0.015, 0.048]
llama-4-scout-17b-16e, patched	0.085	[0.058, 0.103]
GLM-4.5, patched	0.085	[0.061, 0.114]
Qwen3-235B-A22B, baseline (ref.)	0.101	[0.063, 0.134]
mistral-small-4, patched	0.134	[0.096, 0.162]
minimax-m2.1, patched	0.151	[0.123, 0.176]
deepseek-v4-flash, patched	0.192	[0.132, 0.233]
gpt-oss-120b, baseline (ref.)	0.215	[0.173, 0.252]
gemma-4-26B-A4B-it, patched	0.278	[0.215, 0.322]
nemotron-3-super-120b-a12b, patched	0.332	[0.286, 0.371]

D.5 Full-corpus confirmation, across models, in full

§8.5 summarizes this; this is the full table and mechanism comparison. gpt-oss-120b and Qwen3-235B-A22B are the pair the `ir`→`respond` issue was diagnosed on (§8), so both a pre-patch baseline and a post-patch run exist for them. The other ten models were run directly with both patches applied; no pre-patch baseline exists for these ten, and the table below says so rather than leaving the column blank without explanation. GLM-4.5's entry here is the rescored value (its own run-time report read 0.063; `rescore_master.py`, replayed against the current scorer, gives 0.085), the same stale-report correction already noted for nemotron-3-super-120b-a12b.

Table 11: Weighted success rate, same corpus, sorted by the patched column – the one value every model in this table has. Only gpt-oss-120b and Qwen3-235B-A22B were also run before the patch; the other ten were evaluated with both patches already applied, no pre-patch run exists for them.

model	patched	baseline
nemotron-3-super-120b-a12b	**0.332**	no baseline run
gpt-oss-120b	0.310	0.215
gemma-4-26B-A4B-it	0.278	no baseline run
Qwen3-235B-A22B	0.231	0.101
deepseek-v4-flash	0.192	no baseline run
minimax-m2.1	0.151	no baseline run
mistral-small-4	0.134	no baseline run
llama-4-scout-17b-16e	0.085	no baseline run
GLM-4.5	0.085	no baseline run
param-17b	0.041	no baseline run
sarvam-105b	0.028	no baseline run
sarvam-30b	0.028	no baseline run

Both models improve by different mechanisms: gpt-oss-120b's call-match barely moves (0.694 to 0.714) while its `ir→respond` pattern falls to zero, so Patch 2's process checks do most of the work; Qwen3-235B-A22B's call-match moves far more (0.315 to 0.723), matching §8.3's finding that this was its dominant defect – Patch 1 dominates for Qwen3-235B-A22B, Patch 2 for gpt-oss-120b, distinct habits addressed differently per model (these sub-checks predate §D.11's correction; its 12.7%-of-corpus scope makes the effect on these run-wide aggregates small). These are the final reported patched-run numbers for both models. Patch 1's instruction is also close to describing the harness's own scoring mechanics, so we report the improvement as evidence of protocol-specific sensitivity, general agentic improvement, or both: separating them needs transfer to an independently-specified protocol, not more runs on this harness.

D.6 Policy rulebook, in full

§4.4 summarizes the policy signal as the domain's own standing rules, checked against every turn; this subsection gives the rulebook's full structure. A rule shipped inside one episode is an instruction the agent simply follows. A rule given once per domain and never restated has to be *remembered* and *recognised as applying* on the turn where it becomes relevant – that is where the difficulty is, and it is why policy is scored as a signal distinct from call-match: an agent can complete a task's tool calls perfectly and still break a standing rule (§8 works through exactly this).

Table 12: The policy rulebook: 23 checks across four domain rulebooks, all mechanically scored from tool calls, sandbox state, or a concrete target the reply must contain – never a brittle keyword test standing in for a real requirement.

Rule group	What it covers	Checks
P1–P6, P10	UPI payment rules: payee resolution, no invented payee, balance, confirmation match, ambiguous-amount clarification, unknown outcome, no false success.	7
A1–A9	Agri / scheme rules: apply exclusions correctly, don't assert a threshold the reference never gave, don't exceed what a tool returned.	9
L0–L3	Legal rules: answer from the record, no computation the record doesn't support, absence is not disproof, name what is missing.	4
C1–C3	Conversation rules: don't ask for information already given, don't answer before a lookup, don't re-ask an already-answered slot.	3

The counts above describe the rulebook's fixed structure, independent of any one run. `P5` – an ambiguous amount must be clarified, not inferred – is the rule §8's worked example breaks: the agent picks the larger of two candidate amounts and confirms its own guess rather than surfacing the ambiguity. `P6` carries the heaviest weight in the payments rulebook because it is the one a payment tool with only a success/failure interface cannot test.

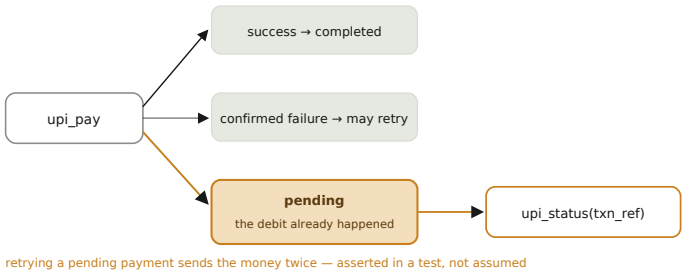


Figure 8: The pending-payment state machine underlying `P6`. A payment returns success, confirmed failure, or pending. Pending means the debit already happened and the outcome is unknown, so it must be resolved with `upi_status` before any retry; only a confirmed failure may be retried. Retrying a pending payment sends the money twice – asserted in a scorer test, not assumed.

D.7 Signal correlation, full matrix

Pairwise Pearson correlation between the four continuous per-episode signal scores, computed over episodes where both signals apply (execution is inapplicable for purely read-only episodes):

Table 13: Signal correlation matrices, baseline runs.

gpt-oss-120b	execution	process	call-match	policy
execution	1.000	0.013	0.243	0.224
process	0.013	1.000	−0.222	0.022
call-match	0.243	−0.222	1.000	0.415
policy	0.224	0.022	0.415	1.000

Qwen3-235B-A22B	execution	process	call-match	policy
execution	1.000	0.099	0.511	0.334
process	0.099	1.000	−0.024	−0.182
call-match	0.511	−0.024	1.000	0.436
policy	0.334	−0.182	0.436	1.000

D.8 Aggregation-rule sensitivity, full table

Table 14: Weighted success rate under the current rule vs. three alternatives, baseline runs (§C.2).

rule	gpt-oss-120b	Qwen3-235B-A22B
current (all 4 signals individually perfect)	0.251	0.081
relaxed conjunction (min of 4 $\geq$ 0.9)	0.255	0.081
arithmetic mean of 4 $\geq$ 0.9	0.506	0.200
geometric mean of 4 $\geq$ 0.9	0.496	0.188

D.9 Full per-language results, ten models, patched

Table 15: Per-language success rate, the original ten models this table was built for. gpt-oss-120b and Qwen3-235B-A22B shown with both patches applied, matching the other eight, which were only ever run patched (§8.5). Nine of ten models peak at `en`; `llama-4-scout-17b-16e` peaks at `ml` (0.144 vs. 0.113 for `en`). `brx` (Bodo) or a similarly low-resource language is lowest for every model. GLM-4.5 and param-17b (Table 11) are not yet broken out by language here; only the aggregate weighted success rate was available for them at submission time.

lang	gpt-oss	Qwen3	gemma	nemotron	deepseek	minimax	mistral	llama4s	sarvam105	sarvam30
as	0.365	0.392	0.351	0.396	0.212	0.225	0.131	0.081	0.036	0.014
bn	0.288	0.342	0.374	0.369	0.225	0.225	0.203	0.081	0.005	0.009
brx	0.095	0.131	0.126	0.090	0.018	0.117	0.068	0.072	0.023	0.009
doi	0.198	0.194	0.257	0.198	0.144	0.167	0.099	0.050	0.018	0.014
en	0.694	0.482	0.509	0.676	0.351	0.423	0.342	0.113	0.086	0.041
gom	0.216	0.216	0.248	0.261	0.171	0.090	0.104	0.050	0.014	0.009
gu	0.387	0.270	0.414	0.491	0.225	0.279	0.221	0.095	0.018	0.009
hi	0.481	0.382	0.370	0.477	0.279	0.233	0.290	0.134	0.069	0.011
kn	0.306	0.293	0.194	0.414	0.149	0.131	0.198	0.099	0.023	0.014
mai	0.333	0.248	0.284	0.257	0.216	0.176	0.230	0.045	0.005	0.009
ml	0.360	0.356	0.401	0.392	0.225	0.198	0.248	0.144	0.027	0.023
mr	0.338	0.252	0.342	0.369	0.158	0.171	0.176	0.077	0.023	0.009
ne	0.288	0.306	0.270	0.333	0.194	0.171	0.126	0.054	0.018	0.023
or	0.378	0.284	0.324	0.306	0.162	0.131	0.068	0.099	0.041	0.014
pa	0.338	0.320	0.396	0.405	0.257	0.185	0.162	0.095	0.032	0.009
sa	0.252	0.207	0.248	0.221	0.189	0.113	0.167	0.054	0.009	0.018
ta	0.351	0.365	0.347	0.446	0.207	0.225	0.261	0.090	0.032	0.009
te	0.405	0.333	0.351	0.414	0.158	0.162	0.275	0.104	0.041	0.005
ur	0.324	0.297	0.342	0.387	0.162	0.203	0.243	0.108	0.027	0.009

The language-level gap is wide for every model (0.005–0.694 across all ten): nine of ten peak at English, and every model's lowest language is Bodo (`brx`) or another low-resource language written in a script shared with a much higher-resource one rather than one with dedicated tooling. We report the full table rather than only a headline "19 languages" count, per the concern that headline coverage can overstate how uniform performance actually is across it.

D.10 Fabrication-check method, in full (Table 16)

§8 summarizes the method behind Table 16; this is the full version. For every `legal_m_*` episode with an `ir→respond` step and no intervening `call`, we extracted every response token matching the benchmark's real document-id pattern (`D` plus digits, e.g. `D1`) and every token matching a plausible but benchmark-external reference pattern (letters, a year, a number, e.g. `AS-2023-0456`), and checked each against that episode's own `scenario.json`. The second pattern is a negative control: we also searched every seed, every language variant, every tool description, and the harness source itself for it, to rule out the model having seen that id pattern anywhere in its context window rather than inventing it outright. Zero matches across the entire corpus and harness source is what makes the 41-instance figure direct evidence of fabrication rather than an inference from the pattern's shape alone.

Table 16: Verifying the fabrication claim in the legal domain (Qwen3-235B-A22B, same 4,258-episode run), against each episode's own ground-truth scenario data.

	value
`ir→respond` instances checked (`legal_m_*` domain)	396
containing a benchmark-external id pattern (e.g. `AS-2023-0456`)	41
that id pattern found anywhere in this corpus's ground truth, tool descriptions, or harness source (i.e. possible leakage)	0
containing the benchmark's real `Dn` id pattern	68
of those `Dn` mentions matching a real id in that episode's own scenario	188 / 188

D.11 The kb_search fix, in full: what changed, and why it moved the two models in opposite directions

§4.4 summarizes the defect and the fix; this is the full account. 19 `kb_search` gold calls, across 6 seeds in the agri and kisan domains, compared a model's search query against a fixed English string regardless of the episode's actual language; we corrected it by routing

the query through the turn-reference mechanism, using translations that already existed and had already passed this corpus's own per-language quality gate. We verified the fix by replaying each affected episode's own already-recorded actions against the corrected episode definition – *no model was called again* – the same way throughout; all numbers in this paper reflect this correction, and all domains outside agri and kisan are numerically identical to their pre-correction values.

The correction moves the two models in opposite directions. gpt-oss-120b's weighted success rate falls slightly, 0.251 to 0.215; Qwen3-235B-A22B's rises, 0.081 to 0.101. Tracing this to the two affected domains specifically (call-match signal score; the strict `all_gold_calls_matched` sub-check moves further in both cases):

Table 17: Before/after the `kb_search` fix, the two affected domains only; all others are unchanged.

	gpt-oss-120b		Qwen3-235B-A22B	
domain	before	after	before	after
agri (call-match)	0.310	0.150	0.140	0.180
kisan (call-match)	0.597	0.587	0.433	0.549

We checked why directly rather than speculate: for every episode whose call-match flipped, we inspected the model's actual recorded `kb_search` query (verbatim examples below). gpt-oss-120b's regression is real, not a fix artifact: in the flipped episodes it queried in English regardless of the episode's language, which the old always-English gold rewarded by construction and the corrected, per-language gold correctly does not – a genuine failure to follow the episode's language in a tool-call argument, not just in its final reply (§9). Qwen3-235B-A22B's improvement is the mirror image: its queries were correct paraphrases in the episode's own language (verified by token-overlap F1 against the reference, above the 0.6 match threshold) that the old English-only gold could not credit. Fixing the localization bug corrected two different scoring errors that happened to point opposite ways for these two models; the net effect on the ranking in Table 6 is that gpt-oss-120b still scores above Qwen3-235B-A22B, but the gap narrows (0.170 to 0.114 at baseline).

D.12 `kb_search` query examples, in full

Two examples of what the fix in §D.11 changes, one per direction:

A query that now correctly fails to match. In a Bengali episode (`agri_season_advice.bn_native.T0`), gpt-oss-120b searched in English – *"chlorophyll central metal atom"* – even though the episode is in Bengali. The old gold was always written in English, so this English query matched it regardless of what language the episode was actually in. The corrected gold is written in the episode's own language, so this same English query now correctly fails to match: the model did not ask its question in Bengali, and the score should reflect that.

A query that now correctly succeeds. In the matched Hindi episode, Qwen3-235B-A22B searched in Devanagari script, transliterated here as *"chlorophyll ke kendra mein kaun sa dhaatu paramaanu hota hai"* – a correct, fully Hindi paraphrase of the reference question. The old English-only gold could not recognize this as a match at all. The corrected, language-matched gold scores it by token-overlap F1 (0.67), which clears the 0.6 match threshold, so the model now gets credit for a query that was correct all along.

D.13 Case-law cross-referencing, in full (§8.2)

The seed's pre-fictionalisation ledger, cross-referenced against public case-law records, identifies it as *M/s. Shriram Investments v. The Commissioner of Income Tax III, Chennai*, 2024 INSC 760, decided 4 October 2024.[1] The case Qwen3-235B-A22B names in its fabricated turn [5] response – *Wipro Finance Ltd. v. Commissioner of Income Tax* – is a real Supreme Court case (Civil Appeal No. 6677 of 2008, decided 12 April 2022),[2] genuinely cited in the

[1] api.sci.gov.in, judgment PDF

[2] indiankanoon.org/doc/98601996

real underlying judgment. The citation the model attaches to that name, *"(2018) 407 ITR 123 (SC)"*, matches neither the real case (decided 2022, not 2018; no case reported under that citation exists in any source we could find) nor the benchmark's own ledger, which records only *"paragraphs 10–11 quoted verbatim"* against the model's claimed *"Paragraph 14"*. The case name was never something the model needed to recall: the scripted user's preceding turn [3] states it verbatim ("Counsel for the appellant's reliance on Wipro Finance Ltd. v. Commissioner of Income Tax..."), drawn directly from the ledger.

D.14 The 2×2 intervention design, in full (§8.4)

Table 18: The 2×2 intervention design, 57-episode slice. All four cells run.

	Patch 2 off	Patch 2 on
Patch 1 off	call-match 0.544, refusal 0.193 (baseline)	call-match 0.451, refusal 0.213
Patch 1 on	call-match 0.983, refusal 0.053	call-match 1.000, refusal 0.526

With all four cells run, the two patches interact rather than add. Patch 2 alone barely moves refusal (0.193 to 0.213) and does not help call-match at all – it actually falls, 0.544 to 0.451, since a model told to say "I don't know" more readily will sometimes say it about a document it could in fact have found. Patch 1 alone is the one that does the call-match work (0.544 to 0.983), but *lowers* refusal to 0.053: the model, now reading documents it had invented, hits the genuine gap more often and still asks the user to fill it in rather than refusing. Only with both patches together do both signals move in the same direction, and past what either manages alone: call-match to 1.000, refusal to 0.526. This resolves the ambiguity a partial design would have left open: Patch 2 is not doing independent work that Patch 1 merely amplifies – on its own it is close to inert, and its effect on refusal only appears once Patch 1 has already fixed what documents the model is reasoning about. The two patches are not two separate fixes that happen to compose; the second is only legible once the first has removed the confound it was otherwise reasoning through.

D.15 Known residual issues, in full

§9 summarizes these; full detail here. **Agricultural domain.** 2 seeds at 2× weight – two scenarios, not domain-wide evidence for either model. **Mandi live-price reproducibility.** The `mandi` domain reads `api.data.gov.in`'s live price endpoint with no snapshot or cache at scoring time; every scored episode hits the live endpoint fresh, while offline tests below the harness's top validation layer replay a fixture instead, so this is the only tool where a scored run and a test run can see different values, and two models scored on different days are not guaranteed to have seen the same price. **Language-drift residue.** An earlier prompt, by naming every language on every episode, primed English-question replies in Devanagari script; naming only the episode's own language fixed most of it, but romanized Hindi (`hi_roman`) still sometimes draws a Devanagari reply, and a few Hindi reading-comprehension episodes answer in English when the question quotes source text verbatim. Measured directly rather than left anecdotal: across the 13 hand-authored and persona-conditioned seeds' `hi_roman` surface text (150 turns), 96.7% are correctly in Roman script; the 5 exceptions (3.3%) are all the same `kb_query` turn, in 5 of 13 seeds, answering a factual question entirely in Devanagari. **`rc_*` relabeling.** The 32 `rc_*` seeds inherited from BhashaBench-Multi's legal split and re-scored here as general document-QA (§B.4) is a re-use note, not a critique of the source: those seeds were correctly labeled under BhashaBench-Multi's own taxonomy, and the mismatch only appears because this benchmark re-purposes the split for a narrower "legal" domain than BhashaBench-Multi's own label meant. **Translation provenance.** Mined content (§B.3) carries its own per-language quality gate (chrF ≥ 65, LLM-judge thresholds) independent of this note. Surface dialogue text for the hand-authored and persona-conditioned seeds (§B.1, §B.2), across the 18 non-English languages, was produced by machine translation from the English originals, without that same chrF/judge gate, since chrF against the English source is not a meaningful metric across a script change – it would read near-zero for a perfect translation into a non-Latin script, and back-translation infrastructure for these two pipelines does not exist yet. What we checked instead, directly and automatically rather than trusting the translation pass: across the 18 languages' surface dialogue for these 13 seeds, 92–96% of

letter characters in every language are in that language's expected script, with no wholesale untranslated English and no wrong-script leakage at scale; the handful of turns below that band are payment handles and account IDs (e.g. `raju@upi`) correctly left untranslated, not mistranslation. This is a script-conformance check, not a fluency or naturalness judgment – it would not catch awkward phrasing a native speaker would flag – but it rules out the grossest failure mode automatically. What a bad translation in these two pipelines cannot do, in any case, is change what gets scored: the gold trace is authored once and shared, unchanged, across every language variant (§4.1), and a build-time test asserts no target-language string appears inside it (§6), so a translation can make a dialogue read awkwardly but cannot make one language's reference answer easier or harder than another's. Native-speaker review of the translated surface text itself is planned as follow-up work and was not complete at submission time; until then, we do not read cross-language differences in these two pipelines as evidence about dialogue naturalness, only about a model's response to a fixed, language-independent task.

E Reproducibility, in full

§7's reproducibility statement points here for exact per-run settings. Every run in this paper – both central models' baseline and patched runs, and the other ten models' patched-only runs – used the identical harness invocation and decoding settings below; models differed only in which weights the endpoint served.

Table 19: Decoding and harness settings, identical across every run in this paper.

setting	value
temperature	0.0 (greedy decoding, no randomness, for repeatability)
max tokens per model call	8,192
max turns per episode (`max_steps`)	24
request concurrency	16
reasoning effort	server default (not overridden)
`enforce_confirm_gate`	off (§4.5)
episodes per run	4,258 (full corpus, every run)

Every model was served behind an OpenAI-compatible chat-completions endpoint (self-hosted vLLM or SGLang, per model), addressed through the harness's `OpenAICompatModel` client; no run in this paper used a commercial API. The harness also supports Anthropic-style model credentials for the L4 live-model smoke test only (Appendix C), which reproduces no number in this paper and exists solely as an end-to-end sanity check. Each model is addressed by the served-model name the endpoint exposes; we report the public checkpoint name throughout (e.g. gpt-oss-120b, Qwen3-235B-A22B) rather than internal endpoint hostnames, which carry no information beyond what the model name already gives and which the supplementary material's anonymization removes in any case.

Re-running any number in this paper requires: the exact corpus and scenario metadata (§5, released as described in the reproducibility statement); the harness and scorer at the version that produced the reported `episodes.jsonl`; and an endpoint serving the named model at the settings above. `rescore_master.py` (Appendix D) replays a run's own recorded actions against the current scorer without calling the model again, and is the fastest way to check a reported number against raw transcripts; running the harness itself against a live endpoint is only needed to check that the transcripts themselves are reproducible, which the L2/L3 test tiers (Appendix C) check on gold traces rather than on live model output.