INTERFACE-INDUCED TRAJECTORY CENSORING: WHEN TOOL-CALL INTERFACES DISTORT EVALUATION AND RL EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-use evaluations typically measure calls through a serving stack, which can report zero calls even when model outputs contain call-shaped payloads. We study interface-induced trajectory censoring: a mismatch between model emissions and the interface that converts them into actions. On selected BFCL v4 subsets, changing only the serving adapter while holding model weights and other evaluation settings fixed moves scores under the official scorer from 0.00 to 0.96 on simple_python and from 0.00 to 0.19 on multi_turn_base. A template–parser factorial experiment shows strong complementarity: neither component replacement alone restores parsing from the baseline, whereas their joint replacement recovers 196/200 first-request parses. Across five Qwen2.5-Coder checkpoints, classifier-positive emissions increase with model size while the mismatched interface reports no calls; a separate matched-interface Qwen3 ladder exhibits few silent emissions through 8B. Experiments on 115 τ-bench retail tasks extend the mechanism to a stateful, non-code environment. In a preregistered, single-seed, 150-step 7B LoRA experiment, changing only the parser registered in verl increases tool executions from 10 to 16,844. However, neither a shared programmatic-feedback evaluation nor a post-hoc common native function-calling evaluation detects a held-out success gain from repair; rescue counts do not increase. Interface repair thus restores tool-mediated experience without a detected performance gain in this setting. These findings show that tool-use measurements depend on the model–interface configuration and motivate layer-wise observability and conformance checks before attributing evaluation or training outcomes to the model alone.

1 INTRODUCTION

Agent benchmarks and training loops rarely consume raw model generations. They consume an action produced by a chain: emission $\rightarrow$ serialization $\rightarrow$ parse $\rightarrow$ execution $\rightarrow$ observation $\rightarrow$ score or reward. A failure at several of these boundaries has the same observable as a model declining to act: HTTP 200 and an empty `tool_calls` array. We call the resulting measurement failure *interface-induced trajectory censoring*. We distinguish masking of emitted actions from constructive suppression: a decoding constraint can change what is emitted by disallowing invalid actions. The latter is an intervention on generation, not evidence that a valid emitted action was censored. Figure 1 shows the layers measured in our experiments.

Historical 1.5B runs motivated the investigation, but their ambiguous zero-execution result does not identify parser masking (Appendix G). We instead organize the evidence around benchmark measurement, component interventions and the formal 7B control of training experience, with explicit limits on what each observation identifies.

The individual Qwen incompatibility was reported previously in vLLM issue #32926 (hanXen, 2026). Our contribution is to quantify it as an evaluation-validity failure, decompose its causes, and follow it into interactive evaluation and RL:

- On BFCL v4, a template–parser 2×2 shows zero recovery from either one-component replacement at the documented baseline and 196/200 parses when both are replaced; on τ-bench, the same joint repair restores 636 executions over 115 tasks.

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.

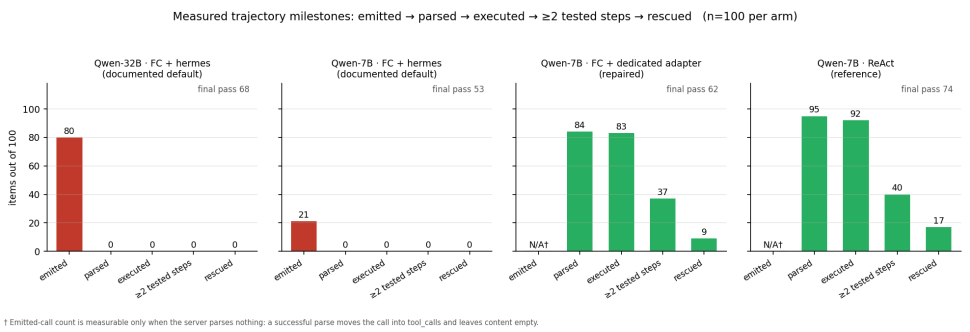


Figure 1: Measured tool-use milestones (n=100 per arm). The fourth bar counts tasks reaching at least two code-bearing tested steps (37 repaired; 40 ReAct), not returned observations. N/A denotes cells that cannot be read from content once the server has moved a successful call into `tool_calls`.

- A five-checkpoint ladder measures a growing absolute undercount under one mismatched interface, while a preregistered matched-interface ladder keeps the silent count at 0–2 per 100.
- A paired Llama intervention shows the opposite interface effect: relative to the official-template baseline, `strict: true` suppresses 22 wrong-tool calls and raises final pass from 44 to 61.
- A preregistered 7B RL control changes only verl's parser registry. Repair restores 16,844 executed and observed calls, but neither the programmatic-feedback nor the post-hoc common native FC evaluation detects a success gain.

The resulting claim is narrower than "the stack matters." A serving interface is part of both the benchmark's measurement function and an agent's effective action space. Consequently, tool-use results require layer-wise observability and a conformance check before model or protocol attribution.

2 Related Work

Evaluation outcomes vary with prompt format, benchmark choice, and other nuisance factors (Sclar et al., 2024; Mizrahi et al., 2024; Dehghani et al., 2021). Tool-use audits similarly expose evaluator defects and ambiguous failure attribution (Bhat et al., 2026; Raj et al., 2026; Soni, 2026; Albayaydh et al., 2026). Those studies generally vary inputs or inspect traces above the serving boundary. We study a component inside the measurement apparatus: an independently configured serialization–parser contract that can erase an action without an error. The closest analogue is a security-log study where reparsing identical bytes changes measured accuracy from 0% to 76% (Garware & Zisad, 2026). We extend that structure to standard agent benchmarks, downstream execution, and RL experience. Harness-Bench argues for reporting model–harness configurations from cross-harness workflow measurements (Yao et al., 2026); we isolate one contract with component-level interventions and follow it into training. Agent-Reactive Bugs finds silent model–harness boundary failures in issue reports (Chen et al., 2026); our evidence is a controlled intervention rather than a bug-report taxonomy. BFCL (Patil et al., 2025), τ-bench (Yao et al., 2025), and other tool benchmarks (Qin et al., 2024; Li et al., 2023) supply the evaluation context; verl (Sheng et al., 2025), GRPO (Shao et al., 2024), and LoRA (Hu et al., 2022) supply the training stack. Constrained generation can hurt reasoning (Tam et al., 2024); our Llama control is a complementary case in which a constraint removes role confusion and improves execution.

3 Framework and Testable Signatures

Let Y be the assistant bytes produced from fixed weights, prompt, and decoding; $C(Y)$ an independent indicator that those bytes contain a tool-intended action; $P(Y)$ the structured action returned by the configured parser; and $E(P(Y))$ the execution and observation produced by the environment.

A benchmark normally records a function of $P(Y)$ or $E(P(Y))$, not Y. We call an item *masked* when $C(Y) = 1$ but $P(Y) = \varnothing$. This definition avoids a metaphysical "true capability": the directly observed fact is that a behavior exists in the raw generation and is absent from the structured trace. It also does not imply a parser bug. The parser may correctly reject bytes outside its grammar; the invalid measurement arises when checkpoint, template, and parser are treated as a compatible bundle without testing their contract.

The framework gives three diagnostic signatures. First, a parser-only intervention cannot change bytes already generated, but can change execution, observations, later turns, and rescues. Our evaluation-time Qwen repair changes parser, template, and few-shot together, so its identical aggregate turn-1 count (53 versus 53) is a consistency check, not layer identification. We instead localize the failure with fixed-output replay and event tracing. Llama's `strict:true` intervention acts earlier by constraining generation and therefore can change turn-1 output.

Second, a size trend in $C(Y) - \mathbf{1}[P(Y) \neq \varnothing]$ requires both changing emissions and a fixed rejection boundary. Across a comparable scale span, our preregistered Qwen3 ladder under a matched envelope does not show the same growth. This supporting cross-family observation does not isolate family from envelope and is not a strict scale counterfactual. Third, in RL the parser changes the sampled experience distribution before a policy-gradient estimator sees it. Restoring actions and observations establishes access to tool-mediated experience, not a measured gradient contribution or eventual learning. The formal 7B control tests the two claims separately: event counts identify experience access, while held-out checkpoints test whether that access changes behavior.

4 Measurement Setup

Our controlled probe uses 100 KodCode problems (Xu et al., 2025), decontaminated against EvalPlus by normalized text/code n-gram containment, paired across arms, with one `run_tests(code)` tool. Function calling uses `tool_choice:auto`, temperature 0, and 2,048 output tokens. We separately run BFCL v4 at pinned Gorilla commit `6ea5797` and all 115 τ-bench `retail` tasks. The latter uses a locally served Llama-3.1-8B user simulator rather than the original harness's `gpt-4o`; its absolute score is therefore not leaderboard-comparable, and we claim only paired arm differences.

Serving uses vLLM 0.27.1 (Kwon et al., 2023). Server-parsed calls are insufficient for the question under study. For the KodCode probe, a shared classifier marks a generation *tight* using a regex for a call-shaped payload naming `run_tests`, a quoted `arguments.code` field and Python-like lexical markers. This heuristic rejects some schema echoes and placeholders but does not validate JSON syntax or Python executability. Against a third-party-adjudicated gold standard its κ is 0.871, and tight-stratum precision is 36/40=0.900. Two annotators achieve $\kappa = 0.967$ on the 62 items presented with identical bytes. Stored evaluation outputs have a 4,000-character cap, which creates false negatives, while 0.900 precision establishes false positives; the net bias is therefore not identified. In the stratified validation sample, cap hits by scale are 0/6, 0/13, 0/15, 0/24, and 10/40 from 1.5B through 32B, so truncation uncertainty is concentrated at the headline rung. Across all 100 stored first outputs per scale, cap hits are 0, 0, 1, 0 and 18; a cap hit does not reveal the omitted continuation. Formal RL rollouts are retained in full with hashes. Full validation appears in Appendix F.

Before interpreting a zero we verify that `tool_choice:required` produces a correctly named, parseable call on the same live server where this control is available; required-mode behavior is not assumed identical to auto-mode parsing. This positive control is distinct from the zero-error admissibility gate; zero auto-mode parses are not an exclusion criterion. An arm enters the clean-arm pass-rate analysis only with 100 items, exit code 0, zero request errors, and matching provenance; this gate is enforced in code. End-to-end analyses are reported separately using full task denominators, including request failures. Appendix A records an earlier invalid $p = 0.648$ that this rule forced us to withdraw.

For paired binary outcomes we use exact two-sided McNemar tests and report both discordant counts; for the formal RL endpoint we report a conservative finite-sample 95% paired-risk interval formed from exact binomial bounds on the two discordance probabilities. It reflects item-pair uncertainty, not seed or inference-retest variation. The historical retrospective family has 27 tests ($0.05/27 \approx 0.00185$); nine later native FC contrasts are separately exploratory (Appendix I). Nei-

ther policy was preregistered. We distinguish a mechanism count (parse, execution, observation) from a task outcome throughout: a large deterministic movement in the former does not license a performance claim when the paired outcome test is null. BFCL scores come from its official scorer; τ-bench outcomes come from final environment state; code-task outcomes execute the generated program against held-out tests. Training events exclude evaluation requests. Historical 1.5B curves exclude two verified EvalPlus defects (`HumanEval/32` and `Mbpp/599`) and use $n = 540$. The preregistered 7B programmatic-feedback evaluation retains all 542 items; both flagged items fail in both arms. The later native FC evaluation scores them differently; Appendix G.15 reports its full denominator and a 540-item sensitivity analysis.

5 RESULTS

5.1 EXTERNAL VALIDITY: BFCL AND τ-BENCH

Using Qwen2.5-Coder-7B-Instruct for both external benchmarks, and holding weights, cases, decoding, seeds, executor, and scorer fixed on BFCL v4, the documented Hermes configuration yields official scores 0.00 on both `simple_python` and `multi_turn_base`; the dedicated Qwen2.5-Coder adapter yields 0.96 and 0.19. Because the adapter changes both template and parser, we ran the full 2×2 (Table 1). Changing either component alone from the documented baseline recovers none (0/200); changing both recovers 196/200. Thus the two one-factor simple effects at that baseline are zero, while the joint replacement shows strong complementarity. This statement concerns parse counts; official BFCL scores were computed for the two end-to-end corner configurations, not all four cells. Each component behaves according to its own contract, but the contracts disagree. The documented arm has zero HTTP errors and 166/200 responses satisfying the BFCL JSON-call criterion despite zero parses. On `multi_turn_base`, repair moves execution from 0 to 98/100 and success from 0 to 19/100. Complete scorer and funnel tables are in Appendix G. "Documented" refers to applying Qwen2.5 family-level guidance, not a Coder-specific guarantee (Appendix D). A new, answer-blind offline JSON extractor applied to the original 100 `simple_python` responses recovers 97 structurally schema-compliant calls; 92 pass the pinned official AST checker without regenerating text (Appendix G.4). This is distinct from the deployed dedicated parser.

Table 1: BFCL first-request parses, template $\times$ parser ($n = 200$).

	documented template	dedicated template
Hermes parser	0	0
Dedicated parser	0	**196**

On paired τ-bench tasks, the same adapter swap moves parsed and executed calls from 0 to 636 and tasks reaching any execution from 0/115 to 103/115. All 1,389 documented-arm requests carry tools; 789 assistant turns contain call-shaped payloads that are not parsed. Replay assigns all 789 turns to the missing-envelope category; none reaches the payload-validation stage, and none is accepted by the configured extractor. Outcomes move from 7 to 10 solved, with three discordant pairs in the repaired direction ($p = 0.25$); we make no outcome claim. This result extends the envelope-layer mechanism to a non-code, stateful environment, but not the scale or RL claims, which remain code-only (Table 2).

Table 2: External benchmarks under documented and repaired adapters. BFCL rows are official scores; τ-bench rows are paired counts over 115 tasks.

measurement	documented	repaired
BFCL `simple_python` score	0.00	**0.96**
BFCL `multi_turn_base` score	0.00	**0.19**
BFCL multi-turn executed/succeeded	0/0	**98/19**
τ-bench parsed/executed calls	0/0	**636/636**
τ-bench tasks reaching a tool	0	**103**
τ-bench tasks solved	7	10

5.2 SCALE UNDER MISMATCHED AND MATCHED INTERFACES

Under the documented Hermes interface, the Qwen2.5-Coder server reports zero calls at all five sizes while tight-classified emissions rise monotonically from 0 to 80/100 (Table 3). A sensitivity estimate is approximately 72 if tight-stratum precision transfers across sizes; this does not restore false negatives or estimate all true calls. A random 300-item replication gives 0, 4, 30, 40, and 81 per 100: the 7B rung moves from 21 to 30, but direction and monotonicity remain. This is a within-family observation under one mismatch, not a scaling law; the server column is constant, so the slope comes entirely from increasingly frequent or regular emissions. The 4,000-character cap creates false negatives, while classifier precision below one creates false positives; the raw count's net bias is not identified.

The preregistered supporting ladder uses Qwen3 checkpoints up to 8B whose required envelope matches Hermes. The prediction "parsed > 0 at every size" was committed before inference. It holds, while the silent count stays 1, 1, 2, and 0 through 8B; the separate mismatched ladder reaches 80 at 32B. This cross-family observation is consistent with contract mismatch, rather than size alone, driving the undercount; it does not isolate the envelope because family and envelope change together. Parsed counts are not monotone, and we disabled thinking because the default exhausted the token budget before a call; both facts were recorded as protocol deviations. An attempted Granite rescue produced no calls under either Hermes or its native parser and is uninformative, not confirmatory.

Table 3: Scale ladders. Tight counts concern unparsed outputs only (including Qwen3); they are not total emitted-call counts. "Silent" denotes these classifier-positive, unparsed outputs.

family/interface	size	parsed	tight, unparsed / 100
Qwen2.5-Coder/Hermes	1.5B	0	0
	3B	0	4
	7B	0	21
	14B	0	36
	32B	0	80
Qwen3/matched	0.6B	86	1
	1.7B	44	1
	4B	70	2
	8B	97	0

A within-lineage Qwen2.5-Instruct control further localizes the effect: under the same template, parser, items, and decoding, parsed rates are 1, 11, 63, 77, and 88 rather than Coder's flat zero. After rerunning with a working executor, multi-turn rescues rise 0, 3, 6, 14, and 23 with that parsed column. The contrast is consistent with checkpoint-specific post-training differences, not parameter count or the shared family/template alone; it does not identify tool-format training as the unique cause. Across 4,254 de-duplicated first turns from all archived arms, replay finds zero cases that vLLM's configured parser should have accepted but did not: this is contract mismatch, not a parser implementation bug (Table 4).

Table 4: Within-lineage Qwen2.5-Instruct control under the same interface and items. Rescues are final passes not passing on turn 1.

size	parsed	turn-1	final	rescues
1.5B	1	13	13	0
3B	11	14	17	3
7B	63	34	40	6
14B	77	38	52	14
32B	88	42	65	23

5.3 LLAMA: A CONSTRAINT SUPPRESSES THE WRONG ACTION

Llama-3.1-8B calls the function requested by the programming problem rather than `run_tests` on 23/100 items. A richer schema leaves 22 wrong calls (discordances 6/5, $p = 1.0$), a thought scaffold worsens them to 59 ($p = 2.84 \times 10^{-10}$), and the official template leaves 22 (1/0, $p = 1.0$). The previously quoted $p = 0.125$ belongs to the final-pass comparison, not wrong-tool labels. Adding `strict:true` to the official-template arm reduces wrong calls from 22 to 0 (22/0, $p = 4.77 \times 10^{-7}$), raises valid execution from 73 to 97, and final pass from 44 to 61, a 17-point single-variable change. The broader terse-FC to official-plus-strict comparison used in the ReAct decomposition is 49 to 61, hence 12 points for the interface bundle ($p = 0.0075$), followed by 19 for protocol ($p = 0.0019$). These are descriptive score differences; neither contrast survives the retrospective family-wise correction. The official-template strict final-pass comparison does survive that correction. Thus the model exhibits role confusion under unconstrained function calling, while the interface determines whether that confusion reaches the environment. Appendix A corrects the corresponding initiation rate from 97 to 74. DeepSeek and Mistral only motivate additional template/token failure layers: the former never injects tools, and all four Mistral FC variants contain request errors, so neither supports a comparable performance claim. Table 5 reports the paired wrong-tool tests and their comparison baselines.

Table 5: Llama-3.1-8B paired controls. The first three interventions do not fix role confusion; `strict:true` does.

arm	wrong-tool calls	exact paired p (baseline)
terse FC	23	—
rich schema	22	1.000 (terse)
thought scaffold	59	2.84×10^{-10} (terse)
official template	22	1.000 (terse)
official template + strict	**0**	4.77×10^{-7} **(official)**

5.4 REPAIR CHANGES EVALUATION TRAJECTORIES

On Qwen2.5-Coder-7B, changing only the evaluation adapter moves parses 0→84, items reaching a second code-bearing tested step 0→37, rescues 0→9, and final pass 53→62. Aggregate turn-1 pass is 53 in both arms, while Llama's schema intervention moves turn-1 pass from 33 to 46 on the official template. Because the Qwen adapter also changes template and few-shot, this aggregate pattern is descriptive rather than a failure-layer locator; fixed-output replay and event tracing provide that localization. The Qwen final gain is not significant at $n = 100$ (16 gains / 7 losses, $p = 0.093$) or after the random $n = 300$ replication and a Bonferroni threshold of approximately 0.00185 ($160 \rightarrow 178$, nominal $p = 0.0385$). Repair restores the mechanism; we do not claim an evaluation outcome effect (Table 6).

Table 6: Qwen2.5-Coder-7B evaluation repair; only the serving adapter changes.

arm	parsed	≥ 2 tested steps	turn-1/final	rescues
FC + Hermes	0	0	53/53	0
FC + dedicated adapter	**84**	**37**	53/62	**9**
ReAct reference	95	40	57/74	17

5.5 PARSER REPAIR RESTORES RL EXPERIENCE; NO HELD-OUT IMPROVEMENT DETECTED

The RL experiment is a preregistered single-variable 150-step 7B GRPO comparison. Both arms are evaluated by the same parser-independent protocol, which automatically executes generated code and supplies test feedback; this is not an evaluation of voluntary tool initiation. Both arms use the same LoRA rank 32, data, mandatory prompt, seed, decoding, verifier, evaluation, and training budget; the sole intervention is verl's registered parser. Full emissions, parser verdicts, executions, observations, and hashes are retained. Table 7 excludes held-out evaluation from training event

counts and uses the same paired 542-item evaluation set at every checkpoint. Both step-0 evaluations use the same base checkpoint and evaluator; their equal aggregate counts are an observation, not evidence from which we infer the evaluation design.

Table 7: Formal 7B RL control. Tight counts generation events; accepted counts parsed calls; exec. counts tool invocations, all observed. Final/rescues count evaluation items.

arm	tight	accepted	exec.	final/rescues, step 0→150
broken FC	8,689	10	10	430→430 / 12→12
repaired FC	17,608	16,912	16,844	430→427 / 12→12

The 16,912 accepted calls occur in 16,844 generation events: 40 generations contain multiple calls and contribute 68 calls beyond the first. The configured one-call cap selects the first parser-returned call and leaves the remaining calls undispatched, not merged. All 16,844 dispatches join to executions and observations. Extraction-to-dispatch counts match per trajectory, not through per-call IDs; Appendix G.10 details this audit boundary.

The broken channel is nearly, not literally, closed: 8,689 tight-classified generation events and only 10 executions are recorded. Repair changes sampled tool experience by more than three orders of magnitude and returns an observation for every execution. Under the common programmatic-feedback evaluation, no held-out improvement is detected: repaired FC ends at 427/542 versus 430/542 for broken FC (paired difference -0.55 percentage points; conservative paired 95% CI $[-1.79, 0.72]$; all three discordances favor broken; exact McNemar $p = 0.25$), and rescues are 12 at both initialization and the endpoint; intermediate checkpoints vary. Relative to its own initialization, repaired FC has one gain and four losses ($p = 0.375$). We do not interpret these non-significant contrasts as harm. At this scale, budget, and single seed, parser repair restores tool-mediated experience, but we detect no improvement in held-out success or rescue counts. Held-out evaluation is identical across arms and deliberately bypasses both training parsers: the current LoRA is queried through verl's native rollout RPC at temperature 0 for up to four 1,024-token programmatic-feedback turns, generated code is executed on the same 542 EvalPlus items, and test output is appended as a user message. Training uses structured tool-role feedback; this format shift may limit transfer, alongside adaptation capacity, training budget and credit assignment. This protocol does not measure native FC initiation. This explains why the broken-training arm can still show 12 evaluation rescues. Each evaluation request attests the loaded global weight step.

A post-hoc common native FC evaluation of all three weights on the same 542 items also detects no repaired-versus-broken success gain: both pass 248/542 versus 239/542 for base (16 discordances each way; difference 0 points, conservative 95% CI $[-3.47, 3.47]$; $p = 1$). Accepted-call initiation occurs on 46/50/47 tasks for base/broken/repaired; tool-test rescues are 2/1/1. Only accepted calls receive tool-role feedback; terminal scoring is separate. This is not an equivalence result or an additional training seed (Appendix G.15).

The repaired job was interrupted after step 116 and resumed from a validated step-90 checkpoint; model, optimizer, scheduler, RNG, dataloader, source, configuration, and weight-step hashes were checked. The repeated step-90 evaluation differs by two final-pass items, reported as test–retest variation. Formal event totals use the final model's lineage—original steps 1–90 plus resumed steps 91–150—and exclude the abandoned 26-update step-91–116 branch. Resume evidence and the available checkpoint endpoints are in Appendix G.10.

Two details prevent common but incorrect readings of this comparison. Both arms use LoRA because 7B full-parameter GRPO does not fit on one 80 GB GPU; this preserves the arm contrast but prevents strict comparison with the historical 1.5B full-parameter curves. Event totals also differ because repair changes trajectories: once calls execute, conversations contain more turns and hence more opportunities to emit another call. We therefore use rates within each arm only to describe its funnel, and use the single registered-parser intervention for causal attribution, not equality of raw denominators. The old ReAct arm with 23,676 executions and flat rescues is directionally consistent, but remains supporting evidence because it changes protocol as well as interface.

6 Implications and Limitations

Measure the funnel. Reports should include emitted-call rate under a stated criterion, parser acceptance, dispatch/execution, observation return, request errors, adapter/template/parser versions and hashes, and task outcome. We provide a 98-line preflight that sends a canonical tools-bearing request, validates the returned name and arguments, and repeats under `tool_choice:required` as a positive control. This screening check flags configurations that fail to return an expected tool call; it does not by itself identify the underlying failure mechanism. Generation constraints are a distinct potential remedy, altering emissions rather than reparsing existing bytes; they do not by themselves ensure task success. This should become a machine-readable conformance report rather than a model-only call rate.

At minimum, such a report should persist model and tokenizer revisions, template and parser hashes, decoding arguments, request errors, the emitted-intent criterion, acceptance, dispatch, execution and observation counts, and the outcome delta. Most fields already exist somewhere in a normal trace. The standard changes the unit of reporting from "model call rate" to a versioned model–interface measurement card, making silent zeros auditable and adapter changes comparable.

Interpretation limits. The scale and RL results use one code task and one tool; only the envelope mechanism is replicated on a non-code benchmark. The scale trend is one family under one mismatch, with single-sample headline arms and a 32B ceiling. The RL result is one 7B LoRA seed and budget, and near-zero samples do not establish zero policy probability. The historical 27-test family has a Bonferroni threshold of 0.00185; only the strongest interventions survive it, and we rest no claim on the nominal or null contrasts. A historical verifier used an unanchored pytest regex and omitted skipped/expected-failure outcomes; the formal arms use the same repaired verifier and retain full rollouts, so that flaw cannot explain their contrast. Serving/training results depend on vLLM 0.27.1 and verl 0.9.0; the later native FC evaluation uses Transformers 5.10.4 and PEFT 0.20.0. Appendix H states all 17 limitations, including the non-random sample, post-treatment conditioning, asymmetric family taxonomy, human-validation scope, provenance corrections, and the observations that would change our conclusions.

Several of those limits bear directly on the headline. The original 100 items are `clean[:100]`, not a random draw; the random 300-item replication preserves the ordering but moves the 7B rung from 21 to 30 per 100. Conditioning protocol comparisons on items parsed by both arms is post-treatment selection, so the Qwen residual of +8.4 points (95% CI $[-0.5, 17.4]$, $p = 0.118$) is unresolved rather than evidence of equivalence. The taxonomy is intentionally asymmetric: Qwen and Llama have controlled repairs, while Mistral has no admissible 100-item FC run and DeepSeek only a template diagnosis. Complete RL attempt-versus-discard attribution exists only for the new 7B runs. The historical 1.5B data remain over-determined and are not used for the causal training claim.

What would change the conclusions. A replicated or longer repaired-FC run with a consistent rescue gain would narrow the scope of the present 150-step, single-seed null result. A second family whose undercount does not grow with size would confine the scale observation to Qwen-plus-Hermes; a comparable rise would strengthen the scaling interpretation. A non-code size ladder would separate the envelope mechanism already seen on τ-bench from the code-shaped payloads used in the current ladder. A second serving stack would identify which findings are vLLM-specific. These tests and decision rules are stated in full in Appendix H, rather than presented as evidence already obtained.

7 Conclusion

A tool-call rate measures a model–serialization–parser–execution composition. On BFCL, neither single-component replacement restores parsing from the baseline, whereas their joint replacement moves the official score from 0.00 to 0.96; in RL, changing only the parser moves observed executions from 10 to 16,844 without a detected gain under either common evaluation protocol. The practical consequence is simple: verify the contract and report each funnel layer before attributing a terminal zero, a protocol gap, or a training curve to the model.

AI USE STATEMENT

Generative AI systems assisted with experiment design feedback, implementation and debugging, result interpretation, literature discovery, and manuscript drafting and editing. The authors inspected the underlying trajectories and logs, reran validation checks, verified every reported number against persisted artifacts, and manually reviewed the final claims and citations. No AI-generated output is treated as experimental evidence. The authors take responsibility for all content, including code, claims, and text produced with AI assistance.

REPRODUCIBILITY STATEMENT

The anonymous supplementary material contains executable statistical code, reduced outcome inputs, component versions and configuration summaries, file hashes, retained preregistration documents, full statistical outputs, reduced per-item RL evaluations, and compact event records including excluded resume segments. Raw BFCL and τ-bench records plus a deterministic RL text sample support content inspection and single-turn BFCL replay. The locally retained full rollout archive is not part of this anonymized package; that disclosure boundary is explicit in its README. Appendix B provides a minimal reproduction path; Appendices D–G provide configurations, arm inventories, validation records, and complete result tables. Appendix A records every known provenance or instrumentation error and its effect on the reported conclusions.

REFERENCES

Wael Albayaydh, Rui Zhao, and Ivan Flechais. Beyond the leaderboard: A synthesis of tool-use, planning, and reasoning failures in large language model agents. *arXiv preprint arXiv:2607.05775*, 2026. Submitted 2026-07-07.

Vishvesh Bhat, Jay Vaghasiya, Muhammad Ahmed Mohsin, and Asad Aali. Benchmarking the benchmarks: A validity audit of tool-calling evaluation. *arXiv preprint arXiv:2607.02577*, 2026. Submitted 2026-06-30.

Jingyi Chen, Songqiang Chen, Hengcheng Zhu, Jialun Cao, Jiasi Shen, and Shing-Chi Cheung. Understanding agent-reactive bugs at the model-harness boundary: An empirical study of llm agent issue reports. *arXiv preprint arXiv:2607.15684*, 2026.

Mostafa Dehghani, Yi Tay, Alexey A. Gritsenko, Zhe Zhao, Neil Houlsby, Fernando Diaz, Donald Metzler, and Oriol Vinyals. The benchmark lottery. *arXiv preprint arXiv:2107.07002*, 2021.

Chaitanya Vilas Garware and Sharif Noor Zisad. When the ruler is broken: Parsing-induced suppression in llm-based security log evaluation. *arXiv preprint arXiv:2605.07293*, 2026. Submitted 2026-05-08.

hanXen. Feature request: dedicated tool parser for qwen2.5-coder models. vLLM issue #32926, opened 2026-01-23, closed as not planned, 2026. https://github.com/vllm-project/vllm/issues/32926.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2023.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. State of what art? a call for multi-prompt llm evaluation. *Transactions of the Association for Computational Linguistics (TACL)*, 12:933–949, 2024.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning (ICML)*, 2025. https://gorilla.cs.berkeley.edu/leaderboard.html.

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. In *International Conference on Learning Representations (ICLR)*, 2024.

Harsh Raj, Vipul Gupta, Anas Mahmoud, Razvan-Gabriel Dumitru, Darvin Yi, Aakash Sabharwal, and Yunzhong He. Model or harness? an interaction-centric taxonomy for localizing agent failures. *arXiv preprint arXiv:2607.28802*, 2026. Submitted 2026-07-30.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design, or: How i learned to start worrying about prompt formatting. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, 2025.

Harsh Soni. Toolfailbench: Diagnosing tool-use failures in llm agents. *arXiv preprint arXiv:2607.04686*, 2026. Submitted 2026-07-06; workshops on Agents in the Wild (AIWILD) and Failure Modes of Agentic AI (FAGEN) at ICML 2026.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on performance of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track (EMNLP Industry)*, pp. 1218–1236, 2024.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, 2025. doi: 10.18653/v1/2025.findings-acl.365.

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

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harnessbench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

A Data Errata

Every number in this paper traces to a trajectory file under `runs/final/`*. This appendix lists the files whose recorded provenance is wrong, the arms that are inadmissible for pass-rate comparison, and what was verified correct. It is reproduced from* `runs/final/ERRATA.md` *in the released repository.*

A.1 SEVEN REACT ARMS RECORD THE WRONG MAX_TOKENS

The following files record `max_tokens=2048` while the generation limit in force was **1024**. Cause: `gen()` in the probe carried a default of `max_tokens=1024` while `gen_fc()` used the global `MAX_TOKENS=2048`, and the provenance written to disk recorded `MAX_TOKENS` for both. Fixed 2026-08-31.

```
traj_v3_Llama8B_react.jsonl        recorded 2048 / actual 1024
traj_v3_Qwen7B_react.jsonl         recorded 2048 / actual 1024
traj_v3_Mistral7B_react.jsonl      recorded 2048 / actual 1024
traj_v11_DS1.3b_react.jsonl        recorded 2048 / actual 1024
traj_v11_DS6.7b_react.jsonl        recorded 2048 / actual 1024
traj_v11_Llama32_1B_react.jsonl    recorded 2048 / actual 1024
traj_v11_Llama32_3B_react.jsonl    recorded 2048 / actual 1024
```

Direction of the bias. The contemporaneous FC arms were genuinely at 2048, so in every affected ReAct-vs-FC comparison **the function-calling side had the larger generation budget**. A larger budget need not improve performance, so this asymmetry alone does not establish the direction of bias. Re-running the three main ReAct arms at a true 2048 changed final pass by at most one item ($80 \to 80$, $74 \to 74$, $32 \to 33$), which is why the correction is small in practice.

Replacement data. The four v11 arms are superseded by `traj_v13_*_react.jsonl`; the three v3 arms by `traj_v14_*_react.jsonl`. All tables in this paper cite the v13/v14 versions; the v3/v11 ReAct arms are retained only as a record.

A.2 LLAMA-3.1-8B INITIATION RATE REQUIRED MANUAL CORRECTION

`traj_v3_Llama8B_fc.jsonl` records an initiation rate of 97/100, but 23 of those calls target **the task function rather than `run_tests`**. The probe did not verify tool names when that batch was collected, so `has_action` did not distinguish them. **The corrected `run_tests` initiation rate is 74/100.**

A re-run restricted to exactly those 23 items under matched configuration (`traj_v8_Llama8B_recheck.jsonl`, seed fixed) reproduced 23/23 as wrong-tool calls, with names such as `can_form_word`, `check_password_strength` and `floyd_warshall`.

Of the same batch, `traj_v3_Qwen7B_fc_nosuffix.jsonl` has an initiation rate of 0 and needs no correction; `traj_v3_Mistral7B_fc.jsonl` has 2, and those two were not individually re-checked for tool name.

A.3 ARMS INADMISSIBLE FOR PASS-RATE COMPARISON

```
traj_v3_Mistral7B_fc.jsonl                  n_err = 2   (rc=2)
traj_v5_Mistral7B_fc_official.jsonl         n_err = 42  (rc=2)
traj_v9_Mistral7B_fc_strict.jsonl           n_err = 3   (rc=2)
traj_v9_Mistral7B_fc_official_strict.jsonl  n_err = 39  (rc=2)
traj_v13_Llama8B_fcstrict_t06_s3.jsonl      n_err = 1   (rc=2)
```

For these arms the **error-rate census remains valid** — that census is itself a result — but these arms are excluded from clean-arm pass-rate comparisons, and no clean-arm p-value is computed for them. Separately reported end-to-end analyses retain the full task denominator, including request failures. An earlier draft computed $p = 0.648$ on one such arm; that number is retracted.

A.4 PROVENANCE SCHEMA DRIFT

Fields were added over the course of the work, giving four schema generations: early arms lack `seed`, `temperature` or `fc_schema`. The fields load-bearing for every claim (`model`, `protocol`, `adapter`, `max_tokens`, `clean_index`) are present in all generations. A `script_sha256` field was never successfully added — the patch was interrupted twice — so script identity across a run is instead guaranteed externally, by hashing the probe before launch (`v13_pinned_hashes.txt`) and re-checking after.

A.5 VERIFIED CORRECT

- **All 50 full-length arms use the identical 100-item set** (`clean[:100]`; unique item-set count = 1), so every cross-arm comparison in this paper is exactly paired.
- Every FC arm ran at a true `max_tokens` of 2048.
- Sampling in the variance experiment is genuine: at `temperature` = 0.6, two seeds differ in first-turn code on 95/100 items and flip outcomes on 16/100.

A.6 THIRD-PARTY COMPONENT VERSIONS

The Qwen2.5-Coder `<tools>` parser is taken from `hanXen/vllm-qwen2.5-coder-tool-parser` (Apache 2.0), downloaded 2026-08-31, at the then-current `main`:

```
commit 1b921501f30cbfe347dccb1db7de3c82a1d55131
  (1b92150, 2026-04-29)
  "fix: buffer partial <tools> prefix in streaming
   to prevent tag leak"

SHA256 (first 20):
  c16bb1f88936a2d96c7c  qwen2_5_coder_tool_parser.py
                       (unmodified)
  736bd175adbf90942c1d  tool_chat_template_qwen2_5_coder.jinja
                       (MODIFIED)
  a95b2a9b91e65b3d452f  tool_chat_template_qwen2_5_coder.jinja.orig
                       (original)
```

The template was modified because its few-shot examples invoke `get_weather`, a tool absent from our schema. Left unchanged it would induce calls to a non-existent tool, simultaneously inflating the initiation count and contaminating the empty-argument statistic. The original is retained alongside.

B REPRODUCIBILITY

The formal 7B configuration and shared held-out protocol are in Appendix D; the checkpoint curve and accepted resume lineage are in Appendix G.10. Appendix I defines the interval and retrospective test family. The anonymous package contains reduced Llama outcomes, 11,952 reduced 7B evaluation records, compact events including excluded segments, lineage metadata and verification scripts. It also includes a small raw-content mechanism archive: 800 first responses from the BFCL factorial, 100 documented and 100 repaired single-turn BFCL records with request schemas and official-scoring inputs, all 230 τ-bench task trajectories, and ten deterministically selected step-1 RL trajectories per arm. The full training rollout archive, original evaluation programs, training logs and heavy checkpoints are not included.

Verification target	Package support
Llama paired tests from reduced item records	Yes
Other historical tests from stated discordances	Yes, counts only
7B checkpoint outcomes and endpoint pairing	Yes
7B recorded funnel and accepted resume segments	Yes
Inspect/replay supplied BFCL raw outputs	Yes
Inspect τ-bench and sampled RL raw contents	Yes
Reclassify every training emission	No
Verify checkpoint weights or raw text hashes	No

From the extracted package root, run:

```
python3 analysis/verify_supplement.py
python3 analysis/submission_statistics.py --output /tmp/stats.json
python3 analysis/bfcl_fixed_replay.py \
  --bundle mechanism_evidence/bfcl
```

The event verifier uses the Python standard library and reconstructs inclusion from segment/step ranges. Dispatch, execution and observation can be uniquely joined. Extraction lacks per-turn

request identity, so accepted-to-dispatch matching is only at trajectory-count granularity. Sampled RL events preserve raw emissions and recorded observations, but omit training task inputs and executable tests: they support text inspection, not independent environment re-execution. BFCL request fields come from its inference log, not a byte-for-byte HTTP request capture. Hashes do not substitute for file contents.

Full-archive-only procedures include `analysis/p3_final_evidence.py`, `analysis/intent.py`, `runs/final/final_table.py`, checkpoint audits and historical figure regeneration. They require data not supplied in the anonymous package and are not its default verification workflow. Historical trajectories and their configuration index reside under `runs/final/` in the full research archive; formal 7B records are retained separately. Only explicitly listed reductions and raw-content subsets are provided in the package.

Model snapshots and serving configurations must be pinned because template inheritance can change with revision. The third-party parser is hanXen (`1b92150`, Apache 2.0); paired few-shot examples were rewritten from `get_weather` to `run_tests`. Original and adapted hashes are retained in the full archive.

The post-hoc native FC evaluation adds `native_fc/outcomes.json` (1,626 reduced item records), `native_fc/statistics.json`, and `analysis/native_fc_results.py`. From the package root run `python3 analysis/native_fc_results.py --outcomes native_fc/outcomes.json --out /tmp/native-fc-recheck`. This reproduces paired contrasts and the timeout inventory; it does not re-execute generated programs. Full raw conversations and GPU loading receipts are retained separately, not promised as part of this reduction.

C Prompts and tool schemas: English renderings

The experiments used Chinese prompts. To keep the submission manuscript entirely in English, this appendix gives faithful English renderings; the byte-exact originals and their hashes are included in the anonymous supplementary artifact. The renderings below were never sent to a model and are provided only for review.

C.1 ReAct prompts

````
REACT_OPTIONAL
--------------
You are a Python coding assistant. You may work in the following format:

Thought: your reasoning
Action: run_tests
Action Input: ```python
<complete code>
```
Observation: (filled in by the system)

Thought: I have confirmed that the code is correct
Final Answer: ```python
<final code>
```

You may submit code for verification with Action, or answer directly.
````

The mandatory variant replaces the final sentence with: "You must first submit code for verification with Action, then give a Final Answer," and marks the loop as repeatable. The main evaluation uses the optional prompt for both protocols. Historical FC–ReAct training and both formal P3 arms use mandatory prompts of matched strength.

C.2 Function-calling prompts

```
FC_OPTIONAL
```

```
-----------
You are a Python coding assistant. Implement the function described by
↪  the user.
You may call run_tests to submit your code to the test runner. It returns
pass/fail status and errors, which you may use to revise and resubmit.

FC_MANDATORY
------------
You are a Python coding assistant. Implement the function described by
↪  the user.
You must first call run_tests to verify your code; do not answer
↪  directly.
The tool returns pass/fail status and errors. If any test fails, revise
↪  the
code and call run_tests again.
```

C.3 TERSE, RICH, AND STRICT SCHEMAS

The terse schema is the documentation-level baseline. Its English rendering is:

```
{"type": "function", "function": {
  "name": "run_tests",
  "description": "Submit the Python code you wrote to the test runner; it
                  returns pass/fail status and errors.",
  "parameters": {"type": "object",
                 "properties": {"code": {"type": "string"}},
                 "required": ["code"]}}}
```

The rich control adds: "This tool is not the function you are being asked to implement. Do not pass the task function's parameters into it. Its only parameter, `code`, is the complete source text you wrote," plus the example `def add(a,b):  return a+b`. Under this warning, 22/100 items still pass the task function's parameters (6/5 discordances, exact $p = 1.000$ versus terse). The strict schema is terse plus `"strict":  true` and `additionalProperties:false`, with strict tool calling enabled. This single change takes the wrong-tool count to zero.

C.4 THOUGHT SCAFFOLD AND ROLE-DISAMBIGUATION CONTROL

The Thought-scaffold prefix reads: "First write your reasoning in a Thought: what the problem asks, its edge cases, and your intended algorithm. Once you have thought it through, call the `run_tests` tool." Nothing else changes; wrong-tool calls rise from 23 to 59 ($p < 0.001$).

The 1.5B role-disambiguation control adds the following contrast to `FC_MANDATORY`:

```
Common mistake: the function requested by the user is not a callable
↪  tool.
The only tool is run_tests, whose sole argument code contains your full
↪  source.

Wrong (task asks for join_integers):
{"name":"join_integers","arguments":{"int_array":[1,2,3]}}

Right:
{"name":"run_tests","arguments":{"code":"def join_integers(int_array):\n
    return ','.join(map(str, int_array))"}}
```

Recoverable call-shaped outputs rise from 52 to 64, but correctly named calls remain zero and final pass falls from 15 to 13.

C.5 INTENT CRITERION

Emission indicators are benchmark-specific and held fixed across arms within each benchmark. For KodCode, the regex below targets `run_tests` and a quoted code payload with Python-like lexical markers. It rejects some schema echoes and placeholders but does not establish JSON validity,

Python executability, or semantic intent. Strong and weak use disjoint trajectory-label sets; tight is computed independently.

For BFCL, the JSON-call indicator detects an embedded decodable JSON object naming a tool offered in the request and containing object-valued arguments, including arguments encoded as a JSON string. Format-marker and JSON-marker indicators are independent broader checks; they overlap and must not be summed. Counts refer to unparsed first-request responses, with at most one count per response for each indicator.

For τ-bench, the unparsed call-shaped indicator detects a generic named-call pattern or a Hermes tool-call marker in assistant messages without structured tool calls. It does not validate membership in the offered tool set or guarantee syntactic validity. KodCode annotation precision is not assumed to transfer to either external benchmark.

```
TIGHT = re.compile(
    r'"name"\s*:\s*"run_tests".{0,200}?"arguments"\s*:\s*\{'
    r'.{0,80}?"code"\s*:\s*"(.{0,4000}?)"\s*\}', re.S)
REAL = re.compile(r'\\n|def |class |return |import |lambda ')

def is_tight_call(raw):
    m = TIGHT.search(raw or "")
    return bool(m and REAL.search(m.group(1)))
```

The regex requires `name` before `arguments` and a closing argument brace after the quoted code field; valid alternative key orders may be missed. KodCode human validation is reported in Appendix F.

D Serving and training configuration

D.1 Per-family serving flags

Baseline provenance and scope. The vLLM 0.27.1 tool-calling documentation, under "Qwen Models", lists `Qwen/Qwen2.5-*` with the `hermes` parser (https://docs.vllm.ai/en/v0.27.1/features/tool_calling/#qwen-models; checked 23 September 2026). Our Coder baseline applies this family-level guidance; the passage does not separately validate a Coder checkpoint. Thus "documented" denotes the origin of the configuration choice, not a Coder-specific compatibility guarantee or evidence of its prevalence in deployment. Issue #32926 already described this mismatch before our experiments. Our contribution is its controlled measurement and downstream consequences, not discovery of the incompatibility or invention of the community repair.

These are the configurations under test, including documented serving flags and the adapter/schema interventions identified in our experiments. The rightmost column summarizes observed limitations or failure symptoms; it is not a uniform flag-omission ablation.

Table 8: Serving configurations and observed symptoms (vLLM 0.27.1).

Family	Flags / intervention	Observed symptom
all	`--enable-auto-tool-choice`	HTTP 400 if omitted
Qwen2.5 (Instruct)	`--tool-call-parser hermes`	1–88/100 parsed across sizes; unparsed outputs include payload failures
Qwen2.5-Coder	Baseline: `hermes`; repair: community parser plus paired template	baseline 0 parsed across sizes; joint adapter 196/200 on BFCL
Llama 3.1 / 3.2	`--tool-call-parser llama3_json` `--chat-template tool_chat_template_llama3.1_json.jinja` plus `"strict": true` in the schema	malformed arguments; without `strict` the terse baseline has 23% wrong-tool calls

Family	Flags / intervention	Observed symptom
Mistral (HF format)	`--tokenizer-mode hf --config-format hf --load-format hf --tool-call-parser mistral --chat-template tool_chat_template_mistral_parallel.jinja` `tool_call_id` must be exactly 9 characters	HTTP 400; the officially recommended parallel template *raises* the error rate from 2% to 42%
DeepSeek-Coder	none available: the chat template never injects `tools`	n/a

Missing auto-tool-choice support and the tested Mistral configurations produce explicit request errors. Other mismatches can return HTTP 200 with empty `tool_calls`, indistinguishable at that field from declining to call a tool. The released 98-line `preflight_toolcall.py` checks a canonical request for a non-empty call with the right name and parseable arguments, then repeats under `tool_choice: required` as a positive control. This is a screening check, not a guarantee of compatibility on all tasks.

D.2 Training configuration

The historical 1.5B runs use the following configuration; they motivate the investigation but are not the formal parser-repair control:

Table 9: Training configuration. All arms share every value below; arms differ only in the protocol and, for the historical runs, the algorithm and seed.

Setting	Value
Model	Qwen2.5-Coder-1.5B-Instruct
Framework	verl 0.9.0 (vLLM 0.27.1 rollout backend)
Algorithm	GRPO ($\times$2 seeds) and RLOO ($\times$1 seed)
Trajectories per step	`train_batch_size` 16 $\times$ `rollout.n` 8 = 128
Steps	150 for the historical runs and formal 7B control; 10 / 3 / 75 for the historical diagnostic arms in Appendix G
Training data	KodCode, decontaminated pool of 7669 items
Held-out evaluation	Multi: 164 HumanEval+, 378 MBPP+; repair: 155 HumanEval+, 299 MBPP+
Denominators	Formal: 542 multi + 454 repair records; historical multi: 540 after excluding two defective items
Agent loop	`tool_agent` (FC) or `react_agent` (ReAct)
Tool	`run_tests`, single `code` argument, sandboxed `pytest`
System prompt	`FC_MANDATORY` / ReAct mandatory — matched strength (Appendix C)
Hardware	single A800 80 GB

The formal control of §5.5 instead uses Qwen2.5-Coder-7B-Instruct, GRPO, LoRA rank and alpha 32 over all linear modules, seed 0, 150 steps, and the same 16 prompts $\times$ 8 rollouts per step in both arms. Both use the mandatory FC prompt and fixed verifier, retain complete rollout text, and evaluate the same 542 paired multi-turn items at steps 0/30/60/90/120/150. Evaluation uses a shared parser-independent programmatic-feedback protocol: the current LoRA is queried through verl's native rollout RPC at temperature 0, for at most four turns and 1,024 tokens per turn; generated code is executed against EvalPlus tests and the observation is appended as a user message. Native RPC metadata must match the expected global weight step. The sole arm difference is `multi_turn.format=hermes` versus the registered `qwen2_5_coder` parser. Full-parameter 7B RL did not fit the single 80 GB card, so both arms use the same parameter-efficient configuration. Logged training settings are learning rate 10^{-6}, temperature 1.0, top-$p = 1$, top-$k = -1$, prompt budget 2,048 and response budget 6,144 tokens, at most five assistant turns, a 1,200-token tool-response cap and one dispatched tool call per generation. These are training budgets, distinct from the four-turn, 1,024-token-per-turn held-out evaluation. Decontamination uses question n-grams 8/5 and solution n-grams 10/6, with a five-gram minimum; containment is intersection divided by the

smaller set, and the clean-ID selection threshold is 0.10. These checks do not rule out semantic overlap or pretraining contamination. This preserves the arm contrast but prevents a strict comparison to the historical full-parameter curves.

D.3 TOOL-CALL INSTRUMENTATION

verl 0.9.0's step metrics contain **no** tool-call count. The metric named `timing_s/agent_loop/tool_calls` is *elapsed seconds*; reading it as a count is a hard error and we flag it because the name invites exactly that.

We therefore instrumented the two agent-side call sites directly, each writing one labelled line per call:

- `CodeTool.execute` — the function-calling path
- `ReActAgentLoop._run_tests` — the ReAct path

The counter must not be placed in `sandbox.run_tests`. The reward function evaluates the final program through that same function, so a counter there conflates reward evaluation with agent tool use. We hit this and moved the counter; the sandbox's own counter was renamed `sandbox_exec_count` so the two can never be confused again.

In the historical diagnostic run, the FC counter file has **0 lines**; the ReAct arm's has **788**, all carrying the `react` label, so the provenance is uncontaminated. Three mutually independent signals agree — `num_turns` pinned at its minimum of 2, tool time exactly 0.00 s, and a dedicated counter at 0.

D.4 CONSTRUCTING THE REPAIRED-FC TRAINING ARM

The control initially appeared blocked at three layers:

1. `multi_turn.format=hermes` selects **verl's own** `ToolParser` registry, which is independent of vLLM's — so installing a vLLM parser plugin has no effect on the training path.
2. A verl-side parser we wrote does recover calls, but **none of them are the tool**: 36/52 target the task function, 16/52 carry executable code elsewhere in the body. Name normalisation cannot repair this, because no call carries code to normalise.
3. The 1.5B checkpoint emits no correctly named call for that parser to accept, and 7B full-parameter training does not fit a single 80 GB card.

The first two observations determine the intervention; the third determines the scale and LoRA configuration. We registered the parser inside verl and ran the matched 7B pair. Thus the earlier statement that no executable repaired-FC arm existed was a description of the configuration surface, not a permanent limitation; §5.5 reports the completed control.

D.5 THE PATH TO THE REPAIRED-FC TRAINING ARM

We encountered four successive obstacles while constructing the control. Each attempt is a measurement, so we retain the record; all were eventually resolved by registering the parser in verl and moving to matched 7B LoRA arms.

(i) A vLLM parser plugin does not reach the training loop. verl's AgentLoop performs its own tool-call parsing (`verl/experimental/agent_loop/tool_parser.py`, registered as `hermes`, adapted from vLLM v0.9.1). `multi_turn.format=hermes` selects *that* parser, not the server's, so attaching a plugin to the vLLM rollout server changes nothing in training — a distinction worth stating because the two look identical from the outside.

(ii) A verl-side parser recovers calls, but none of them are the tool. We implemented a `qwen2_5_coder` parser inside verl's registry accepting `<tools>` tags, `json` fenced blocks and bare JSON. In the historical 1.5B diagnostic with a recorded mandatory-prompt file override (n=100), it lifts recoverable calls from **0/100 (hermes) to 52/100**. The original override file is unavailable locally, so exact prompt-byte equivalence to training cannot be verified (Appendix E). Of those 52, **zero name** `run_tests` (Table 31). Every call targets

the function the task asks the model to *write* — `can_form_word({"tiles", "word"})`, `check_password_strength({"password"})` — the same signature as Llama's failure in §5.3, now at a 5× smaller model. **Name normalisation cannot repair this: no call carries code to normalise.**

(iii) A role-disambiguation few-shot makes it worse, not better. Since the failure is semantic rather than syntactic, we targeted it directly: a system prompt containing the wrong call and the right call side by side, with an explicit warning that the task function is not a tool. Same model, same 100 items, same server; only the system prompt changes. The intervention made the model *more fluent at emitting the wrong call* (52 → 64 recoverable) and slightly worse at the task (15 → 13), with the count that matters staying at zero (Table 32). This is the third independent instance of the pattern in §G.9.

(iv) Constrained decoding is unavailable in this stack. Forcing the tool choice, or schema-constrained decoding, is not exposed by verl 0.9.0's vLLM rollout path (its only `strict` key belongs to the profiler). This is a limitation of the tested rollout path, not an argument against constrained decoding. Parser repair accepts existing emissions, whereas generation constraints alter the output distribution. The latter is a distinct potential remedy for serialization or allowed-name violations; neither remedy by itself establishes correct arguments or successful task behavior.

E ARM INVENTORY

Every full-length arm, generated from the trajectory files' own provenance rather than transcribed by hand. All 50 arms share one item set (`clean[:100]`; distinct-set count verified = 1), so every cross-arm comparison in the paper is exactly paired. Fields recorded as `---` were absent from that generation's provenance schema (Appendix A.4); this is disclosed rather than back-filled. Seven arms carry a known-wrong `max_tokens` record and are listed in Appendix A.1; five are inadmissible for pass rates and are listed in Appendix A.3.

The `strength` column is the recorded CLI selector, not necessarily the effective prompt. For `a14` and `roleprobe`, records specify `sys_file`; the archived probe loads this file after selecting the built-in prompt, overriding `strength=optional`. The retained role-disambiguation file explicitly mandates `run_tests`. The original `sys_fc_mandatory.txt` file was not found in the local archive, so the a14 run's exact prompt bytes cannot be independently verified; its name alone is not proof. The zero correctly named calls remain an observed outcome, not a fully byte-matched prompt replication.

Not listed: `v8_Llama8B_recheck` (23 items, a targeted qualitative re-run) and eight `*smoke*` files at n=3. Neither is a formal arm.

Table 10: All 50 full-length arms and their recorded configuration.

Arm	Model	Prot.	Strength	Schema	Adapter	tok	temp	seed
a14_Qwen1.5B_fc_mandatory	Qwen-1.5B	fc	optional	terse	cross_family	2048	—	—
roleprobe_Qwen1.5B_fc_roledisambig	Qwen-1.5B	fc	optional	terse	cross_family	2048	0.0	0
v11_DS1.3b_react	DS-1.3B	react	optional	terse	cross_family	2048	—	—
v11_DS6.7b_react	DS-6.7B	react	optional	terse	cross_family	2048	0.0	—
v11_Llama32_1B_fc_strict	Llama-3.2-1B	fc	optional	strict	cross_family	2048	0.0	—
v11_Llama32_1B_react	Llama-3.2-1B	react	optional	terse	cross_family	2048	0.0	—
v11_Llama32_3B_fc_strict	Llama-3.2-3B	fc	optional	strict	cross_family	2048	0.0	—
v11_Llama32_3B_react	Llama-3.2-3B	react	optional	terse	cross_family	2048	0.0	—
v13_DS1.3b_react	DS-1.3B	react	optional	terse	cross_family	2048	0.0	—
v13_DS6.7b_react	DS-6.7B	react	optional	terse	cross_family	2048	0.0	—
v13_Llama32_1B_react	Llama-3.2-1B	react	optional	terse	cross_family	2048	0.0	—
v13_Llama32_3B_react	Llama-3.2-3B	react	optional	terse	cross_family	2048	0.0	—
v13_Llama8B_fcstrict_t06_s1	Llama-3.1-8B-Instruct	fc	optional	strict	cross_family	2048	0.6	1
v13_Llama8B_fcstrict_t06_s2	Llama-3.1-8B-Instruct	fc	optional	strict	cross_family	2048	0.6	2
v13_Llama8B_fcstrict_t06_s3	Llama-3.1-8B-Instruct	fc	optional	strict	cross_family	2048	0.6	3
v13_Llama8B_react_t06_s1	Llama-3.1-8B-Instruct	react	optional	terse	cross_family	2048	0.6	1
v13_Llama8B_react_t06_s2	Llama-3.1-8B-Instruct	react	optional	terse	cross_family	2048	0.6	2
v13_Llama8B_react_t06_s3	Llama-3.1-8B-Instruct	react	optional	terse	cross_family	2048	0.6	3
v14_Llama8B_react	Llama-3.1-8B-Instruct	react	optional	terse	cross_family	2048	0.0	—
v14_Mistral7B_react	Mistral-7B	react	optional	terse	cross_family	2048	0.0	—
v14_Qwen7B_react	Qwen-7B	react	optional	terse	cross_family	2048	0.0	—
v3_Llama8B_fc	Llama-3.1-8B-Instruct	fc	optional	—	cross_family	2048	—	—
v3_Llama8B_fc_legacy	Llama-3.1-8B-Instruct	fc	optional	—	cross_family	2048	—	—

Arm	Model	Prot.	Strength	Schema	Adapter	tok	temp	seed
v3_Llama8B_react	Llama-3.1-8B-Instruct	react	optional	—	cross_family	2048	—	—
v3_Mistral7B_fc	Mistral-7B	fc	optional	terse	cross_family	2048	—	—
v3_Mistral7B_react	Mistral-7B	react	optional	terse	cross_family	2048	—	—
v3_Qwen7B_fc_legacy	Qwen-7B	fc	optional	terse	cross_family	2048	—	—
v3_Qwen7B_fc_nosuffix	Qwen-7B	fc	optional	terse	cross_family	2048	—	—
v3_Qwen7B_react	Qwen-7B	react	optional	terse	cross_family	2048	—	—
v4_Llama8B_fc_cot	Llama-3.1-8B-Instruct	fc	optional	terse	cross_family	2048	—	—
v4_Llama8B_fc_rich	Llama-3.1-8B-Instruct	fc	optional	rich	cross_family	2048	—	—
v5_Llama8B_fc_official	Llama-3.1-8B-Instruct	fc	optional	terse	cross_family	2048	—	—
v5_Mistral7B_fc_official	Mistral-7B	fc	optional	terse	cross_family	2048	—	—
v5_Qwen1.5B_fc_intent	Qwen-1.5B	fc	optional	terse	cross_family	2048	—	—
v5_Qwen14B_fc_intent	Qwen-14B	fc	optional	terse	cross_family	2048	—	—
v5_Qwen32B_fc_intent	Qwen-32B	fc	optional	terse	cross_family	2048	—	—
v5_Qwen3B_fc_intent	Qwen-3B	fc	optional	terse	cross_family	2048	—	—
v5_Qwen7B_fc_intent	Qwen-7B	fc	optional	terse	cross_family	2048	—	—
v6_Llama8B_fc_strict	Llama-3.1-8B-Instruct	fc	optional	strict	cross_family	2048	—	—
v6b_Qwen1.5B_fc_plugin	Qwen-1.5B	fc	optional	terse	cross_family	2048	—	—
v6b_Qwen3B_fc_plugin	Qwen-3B	fc	optional	terse	cross_family	2048	—	—
v6b_Qwen7B_fc_plugin	Qwen-7B	fc	optional	terse	cross_family	2048	—	—
v9_Mistral7B_fc_official_strict	Mistral-7B	fc	optional	strict	cross_family	2048	—	—
v9_Mistral7B_fc_strict	Mistral-7B	fc	optional	strict	cross_family	2048	—	—
x2_DS1.3B_react	DS-1.3B	react	optional	—	—	—	—	—
x2_DS6.7B_react	DS-6.7B	react	optional	—	—	—	—	—
x2_Llama8B_fc	Llama-3.1-8B-Instruct	fc	optional	—	—	—	—	—
x2_Llama8B_react	Llama-3.1-8B-Instruct	react	optional	—	—	—	—	—
x2_Mistral7B_fc	Mistral-7B	fc	optional	—	—	—	—	—
x2_Mistral7B_react	Mistral-7B	react	optional	—	—	—	—	—

F HUMAN VALIDATION OF THE INTENT CRITERION, IN FULL

The scale trend in §5.2 rests on an automated classifier, so we validated it against human judgement. 98 outputs were sampled **stratified by classifier verdict** (40 `tight`, 28 `strong`-but-not-`tight`, 30 neither) — deliberately over-sampling the decision boundary so both false positives and false negatives are estimable. Sampling is seeded and the annotator saw only the raw output, never the classifier's label. We report **two rounds**, because the first is itself a finding.

Table 11: Human validation of the intent criterion, round 1 and its adjudication. The classifier is used only to count what the server discarded, so its recall matters more than its precision.

	agreement	Cohen's κ	precision	recall	FP	FN
Round 1 (raw output only)	86.7%	0.713	70.0%	96.6%	12	1
After adjudication	**96.9%**	**0.936**	**95.0%**	97.4%	2	1

The 12 round-1 false positives split into two causes, and the second is ours, not the annotator's. **Eleven are reading-order failures**: the tool call sits mid-document, after a fenced ```` ```python ```` block (match positions 1209–2585 in outputs of median length ∼2000); the annotator read the code block, concluded "this is a direct answer, not a call", and stopped. **One is an instrument failure**: for a 32B item the matched span begins at character 2585 and the annotation pack truncated every output at 2400, so the evidence was *not in the pack at all* and that item could not have been labelled correctly. Re-shown the matched span alone, 10 of 13 disputed items were reversed. The corrected pack carries every output in full.

The first adjudication was one-sided, and we do not treat its κ as a reliability estimate. Auditing the procedure after the fact: the 13 adjudicated items were selected *exactly* as the items where the annotator disagreed with the classifier (13/13 overlap), and all 10 reversals moved *toward* the classifier, none away. Under that selection rule agreement can only rise, so $0.713 \rightarrow 0.936$ is a property of which items were re-examined, not evidence of reliability. We report the second annotator instead.

A SECOND INDEPENDENT ANNOTATOR

A second annotator, blind to the classifier, to the first annotator's labels and to the first pack's item order, labelled the identical 98 items under a verbatim-identical rubric. Two changes were made to

the instrument: outputs are carried **in full**, and the adjudication rule was fixed in advance to review *all* three-way disagreements rather than only those disagreeing with the classifier.

Table 12: Inter-annotator reliability. Of 98 sampled items, 97 have complete paired A1/A2 records. The identical-byte subset removes a presentation confound introduced by the first pack's 2,400-character cap.

Cohen's κ (95% CI, bootstrap)	all items (n=97)	**identical bytes (n=62)**
A1 vs. A2 (inter-annotator)	0.736 [0.595, 0.866]	**0.967 [0.893, 1.000]**
A2 vs. classifier	0.936 [0.852, 1.000]	**0.967 [0.896, 1.000]**
A1 vs. classifier	0.712 [0.564, 0.849]	0.934 [0.829, 1.000]

Of the 12 paired disagreements, 11 have full text exceeding 2,400 characters and A1 saw a different presentation from A2; the first-round audit attributed most errors to reading order and one to evidence actually omitted by truncation. We therefore do not assign all 11 to one cause. Repeated A1–A2 intervals use 5,000 bootstrap resamples, seed 7 and sorted item IDs; earlier differing bounds reflected inconsistent bootstrap inputs/order, not different annotators. On the 62 items where both read identical bytes the two agree on 61, and **inter-annotator κ = 0.967**.

Third-party adjudication, and the final labels. Fifteen items were not unanimous across A1, A2 and the classifier. All fifteen went to a third adjudicator under the pre-registered rule — selected by *any* three-way disagreement, never by disagreement with the classifier — who saw full text and no one else's labels. The adjudicator sided with A2 on 11, with the classifier on 9, and with A1 on 6, and **moved away from the classifier on 6 of 15**. That last number is the check that matters: the first round's adjudication moved away from the classifier zero times out of thirteen, which is what an outcome looks like when the selection rule has already decided it. Combining the 83 unanimous items with the 15 adjudicated ones gives a 98-item gold standard. The third adjudicator was blind to classifier and annotator labels; the classifier nevertheless affected the original stratified sample and which non-unanimous items required adjudication, so we do not call the gold set classifier-independent.

Table 13: Validation against the adjudicated gold standard. Pairwise denominators differ because one sampled item lacks a complete A1/A2 pair; classifier–gold uses all 98 items.

Cohen's κ (95% CI, bootstrap)	full n	full sample	identical bytes (n=62)
A1 vs. A2 (inter-annotator)	97	0.736 [0.595, 0.866]	**0.967 [0.893, 1.000]**
Classifier vs. gold standard	98	**0.871 [0.756, 0.958]**	0.934 [0.829, 1.000]

Sensitivity estimate, not a corrected population total. Tight-stratum precision is 36/40=0.900. If this precision transfers to the 32B arm, multiplying 80 by 0.900 gives approximately 72 classifier-positive outputs expected to satisfy the gold criterion. This does not restore false negatives and is not an estimate of all true calls. The stratified gold sample contains 4 false positives and 2 false negatives; their unweighted difference is not a population bias estimate. Uniformly applying 0.900 yields $0, 3.6, 18.9, 32.4, 72$ by construction, but scale-specific precision is unmeasured, so this calculation does not independently validate the trend.

For the record, this number moved twice as the validation improved, and we report the trajectory rather than only its endpoint: **0.950** under the first, one-sided adjudication; **0.975** under A2's labels alone; **0.900** against the adjudicated gold standard. The first was inflated by its selection rule, the second rested on a single rater, and only the third is built from a rule fixed before the labels were seen.

A bound on the measurement itself. The probe stores at most 4000 characters of raw output per item; 10 of the 98 sampled outputs reach that ceiling, and one grey-zone item is a call truncated by it. The classifier and both annotators read identical bytes, which is what agreement requires, but a call emitted beyond character 4000 is invisible to all three. All ten cap hits occur in the 32B subset of this stratified sample: by increasing scale the hit counts are 0/6, 0/13, 0/15, 0/24, and 10/40. These are sample diagnostics, not population truncation rates, but they show that the uncertainty is concentrated at the headline rung. The cap therefore creates downward pressure through false negatives. Because the classifier also has false positives (tight precision 0.900), however, the net bias

of the raw count is not known; we report both the uncorrected count and the assumption-dependent sensitivity estimate rather than calling the total a strict lower bound.

G SUPPLEMENTARY RESULTS

G.1 HISTORICAL OBSERVATIONS THAT MOTIVATED THE STUDY

Historical 1.5B runs showed numerical pass gains of 2.6–2.8 points ($p = 0.064/0.078/0.077$), not statistically significant. Of newly passing items, 91–94% passed on turn 1; multi-turn rescues ranged from 5 to 10 out of 540 across the reported checkpoint sequences. These small exploratory counts prompted the investigation but do not identify an interface effect or independently establish absence of learning.

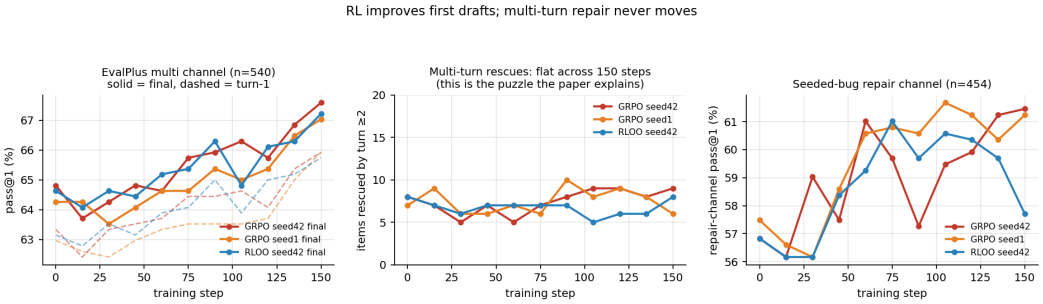


Figure 2: Left: pass@1 rises over 150 steps on all three runs, driven by first drafts (dashed) rather than final answers. Centre: multi-turn rescues range from 5–10 out of 540 across checkpoints, algorithms and seeds. Right: the seeded-bug repair channel does improve, confirming the model is learning something about fixing code; no clear increase in multi-turn rescue counts was detected in these historical runs. §5.5 shows the tool was never successfully called in any of these runs.

G.2 RESULT TABLES REFERENCED FROM §5

Table 14: BFCL v4, first HTTP request per case. Emission indicators are measured only where parsing failed; JSON-call, format-marker and JSON-marker checks overlap and must not be summed.

Arm	cases	HTTP err	first parsed	unparsed	JSON-call	format	JSON
`hermes` (documented)	200	0	**0**	200	166	169	187
`qwen2_5_coder` (repaired)	200	0	**196**	4	0	1	2

Table 15: τ-bench `retail`, all 115 tasks paired across arms (none dropped).

	documented	repaired
assistant turns	1389	1971
server-parsed calls	**0**	**636**
emitted but unparsed	**789**	6
tool executions / observations returned	**0**	**636**
tasks entering the tool loop at all	**0**	**103**
tasks solved	7	10

Table 16: Same probe, items and criterion under a *matched* envelope.

Size	server-parsed	**silent count / 100 (tight, unparsed)**
Qwen3-0.6B	86	**1**
Qwen3-1.7B	44	**1**
Qwen3-4B	70	**2**
Qwen3-8B	97	**0**
Qwen2.5-Coder 1.5B–32B (mismatched)	0 at every size	**0** → **4** → **21** → **36** → **80**

Table 17: Qwen2.5-Instruct under the identical mismatched configuration, with both ladders' task outcomes measured on a working executor.

	Instruct: parsed / unparsed indicators			Coder		Instruct		
Size	parsed	**tight**	weak	parsed	turn-1 = final	turn-1	final	**rescued**
1.5B	1	0	7	0	31	13	13	**0**
3B	11	56	2	0	37	14	17	**+3**
7B	63	21	0	0	53	34	40	**+6**
14B	77	19	0	0	53	38	52	**+14**
32B	**88**	8	0	0	68	42	65	**+23**

Table 18: Llama-3.1-8B arms, paired on the same 100 items.

Arm	Calls issued	Wrong-tool	Executed	Turn-1	Final
terse (parser only)	97	23	74	34	49
+ official template	95	22	73	33	44
+ official template **+ `strict`**	98	**0**	**97**	**46**	**61**
ReAct (reference)	98	0	97	**61**	80

Table 19: Decomposing the ReAct–FC gap on Llama, paired McNemar.

Comparison	b/c	p	Descriptive difference
terse-FC → strict-FC (49 → 61)	15/3	0.0075	**interface repair, +12**
strict-FC → ReAct (61 → 80)	27/8	0.0019	**protocol, +19**
terse-FC → ReAct (49 → 80)	38/7	3.1e-06	both, +31

Table 20: Repair loop on Qwen2.5-Coder-7B. Tested steps count generations with extracted code sent to the sandbox, not all assistant outputs. The historical ReAct row uses v3; the unified v14 rerun has mean tested steps 1.93 with unchanged pass/rescue counts.

Arm	Parsed	Mean tested steps	≥2 tested steps	Turn-1	Final	Rescued
ReAct	95	1.92	40	57	**74**	17
FC + hermes	**0**	0.96	**0**	53	53	**0**
FC + dedicated adapter	**84**	1.82	**37**	53	**62**	**9**

Table 21: RL training rollouts, matched in model, data, hyper-parameters, seed and prompt strength.

Arm	Steps	`num_turns/mean`	Tool time	Tool calls / rollout
FC (`tool_agent` + hermes)	10	**2.000** (constant)	**0.00 s**	**0.000**
ReAct (`react_agent`)	3	5.883	12.21 s	**2.052**

G.3 Offline re-parse matrix and failure-layer decomposition

To exclude "your extractor is simply better than `hermes`", we re-parsed the *same stored bytes* under four rules (`analysis/reparse_matrix.py`, no GPU).

Table 22: The same bytes, re-parsed offline under four rules. Where the server already parsed a call, `content` is empty and offline replay has nothing to read; those cells are N/A rather than zero. The gap is a parser mismatch, not a claim that no parser could have read the output.

Arm (server-parsed = 0)	server	hermes replay	`<tools>` replay	bare JSON	tight
Qwen-1.5B	0	0	0	0	0
Qwen-3B	0	0	0	5	4
Qwen-7B	0	0	0	22	21
Qwen-14B	0	0	0	42	36
Qwen-32B	0	**0**	0	**100**	**80**

The `hermes` column is zero at every scale: the model never produces that format. The bare-JSON and tight columns are not: the calls exist, and their difference from the server column is precisely what the format mismatch removes. For arms where the server *did* parse, offline content-only re-parsing is **undefined, not zero** — a successful parse moves the call into `tool_calls` and leaves `content` empty.

Replaying vLLM 0.27.1's `hermes` extractor line for line over the stored bytes (`analysis/failure_layer.py`) separates the layer at which each item is lost: no `<tool_call>` envelope at all; envelope present but the JSON payload malformed; payload well-formed and rejected only because `json.loads` does not tolerate trailing text in the capture; or a genuine parser loss, meaning the replay succeeds where the server did not.

Table 23: Where the unparsed items are lost. These are first-failure categories of all unparsed outputs, not counts of identified tool attempts. **Genuine parser loss is zero in every arm** — and across all 4254 de-duplicated first turns in the archive.

Arm	no envelope	payload malformed	strictness only	**parser loss**
Coder 1.5B–32B (each size)	100	0	0	**0**
Instruct 1.5B	93	6	0	**0**
Instruct 3B	22	41	26	**0**
Instruct 7B	3	33	1	**0**
Instruct 14B	13	6	4	**0**
Instruct 32B	1	9	2	**0**

Missing envelopes alone do not identify censoring: the 100 Coder outputs at each size include both call-shaped and non-call outputs (notably at 1.5B). A separate emission criterion is needed to identify candidate masking. Instruct outputs reaching the payload check may break JSON, almost always by embedding Python whose docstring quotes or raw newlines are left unescaped inside the `code` string. And **in no arm — across the whole archive — did the server fail to parse a call that its own extractor would have accepted.** "Censoring" throughout this paper therefore means *the attempt never reaches the parser in a form it is defined to accept*, not *the parser discards valid calls*.

The repair-loop table uses the stored `n_turns` counter, which advances only when code is extracted and tested. The Hermes arm has 100 first assistant outputs but four yield no code, hence 96/100 tested steps. Its mean assistant-output count is 1.00; the adapter's is 1.90, rather than its 1.82 tested-step mean. Reaching a second tested step is stricter than merely receiving feedback: a subsequent unextractable response does not increment this counter.

G.4 BFCL: GATING AND THE FULL REPRODUCTION RECORD

Fixed-output replay on single-turn cases. We added a post-hoc, answer-blind replay of all 100 `simple_python` documented-arm responses from the clean run. A deterministic extractor scans left to right for maximal JSON objects or arrays, accepts named calls with object arguments (including JSON-encoded argument strings), and retains all recovered calls. It neither repairs malformed JSON nor selects a call using the answer key. All 97 recovered responses name an offered tool and pass structural signature checks (required fields, types, declared parameter names, array items and enums). Format annotations are not validated. Of the 100 original responses, 92 pass the official BFCL AST checker at the experiment's pinned commit. The 100 repaired-arm outputs reproduce

96 official successes as a positive control. Scoring code is unchanged; the offline loader supplies the same `underscore_to_dot=True` registry property without importing online inference SDKs.

This is an additional offline interpretation rule, not the community dedicated parser: none of the original 100 responses contains its required `<tools>` start tag. The result demonstrates that correct single-turn calls can be recovered from fixed emitted bytes. It does not attribute the remaining difference to template changes alone, establish a safe general-purpose fallback parser, or recover a counterfactual multi-turn success rate. Changing a multi-turn action would change subsequent environment feedback and generations. The supplement includes raw responses, the recorded request schemas, answer keys, per-item verdicts and the pinned checker sources; see Appendix B.

Each arm passes a two-case end-to-end smoke test before its full run, and a run is rejected unless the result count, the score file's `total_count` and the number of distinct case ids in the recording proxy all agree at 200 with zero HTTP errors. Both arms run at `max_model_len = 16384`, `temperature = 0` and inference seed 0, on cases drawn at manifest seed 20260901. Registration of BFCL's OpenAI handler happens in memory at import time, so `bfcl_eval`'s own files are left byte-identical; a recording proxy sits between benchmark and server and logs every request.

We ran the pair twice. The `hermes` arm is identical across runs in every column: 0/200 parsed, 166 tight, 480 requests, 0 errors. The repaired arm moves by one case between runs — **197/200 versus 196/200** first parsed, **99/100 versus 98/100** executed — while success is 19 in both. Greedy decoding does not make vLLM bitwise deterministic across differing batch compositions, and this arm runs eight threads through a multi-turn executor, so we report its intermediate funnel columns as ± 1 rather than exact.

Table 24: BFCL `multi_turn_base`, n = 100, joined to the HTTP trace by case id.

Arm	tight (unparsed)	first parsed	any parsed	executed	**success**	rescued
`hermes`	69	0	0	0	**0**	0
`qwen2_5_coder`	0	97	98	98	**19**	0

Table 25: Two quantified failure layers; the full four-family table is Table 26.

Family	Layer	Symptom	Detectable in advance?	Remedy
Qwen2.5-Coder	parser	emits a fenced JSON block, not `<tool_call>`	× template check **false-positives**	dedicated adapter
Llama-3.1-8B	schema	calls the *task function* as a tool (23%)	× requires argument inspection	`strict: true`

G.5 The full four-family taxonomy

Table 26: Four serving diagnostics, not an exhaustive taxonomy. Only Mistral's failure announces itself.

Family	Layer	Symptom	Detectable in advance?	Remedy
DeepSeek-Coder	template	chat template never injects tools	✓ template inspection	none
Qwen2.5-Coder	parser	emits a fenced JSON block, not `<tool_call>`	× template check **false-positives**	dedicated adapter
Llama-3.1-8B	schema	calls the *task function* as a tool (23%)	× requires argument inspection	`strict: true`
Mistral-7B-v0.3	token	repeats `[TOOL_CALLS]` → HTTP 400	∼ errors, but only sometimes	**none found**

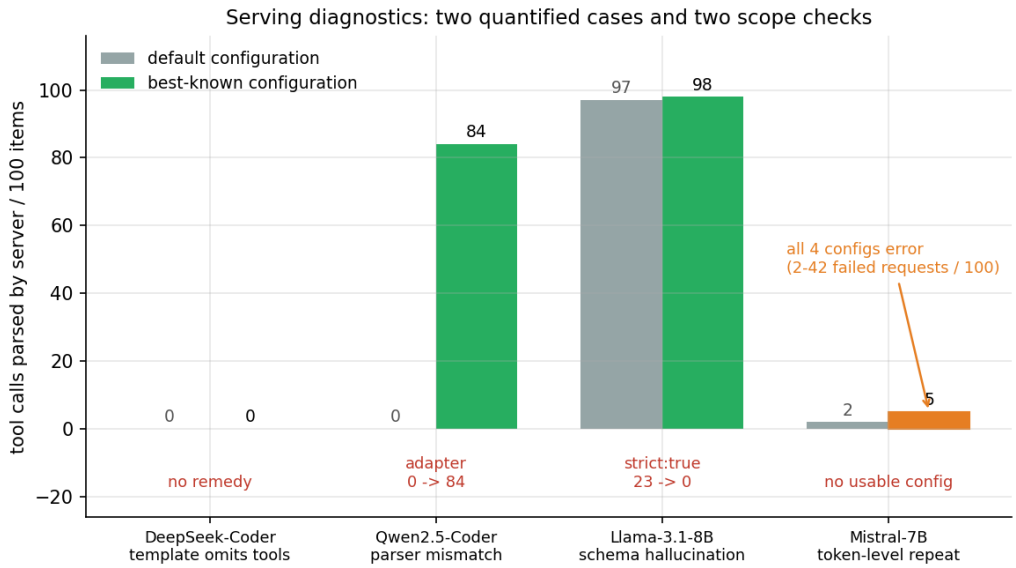


Figure 3: Serving diagnostics with a consistent server-parsed metric. Llama is 97 versus 98 parsed; wrong-tool and executed counts are reported separately. Grey is the configuration a reader arrives at by following the documentation; green is the best configuration we found. Mistral never produces an error-free run under any of four configurations, so its bars are not comparable with the others and are drawn in orange.

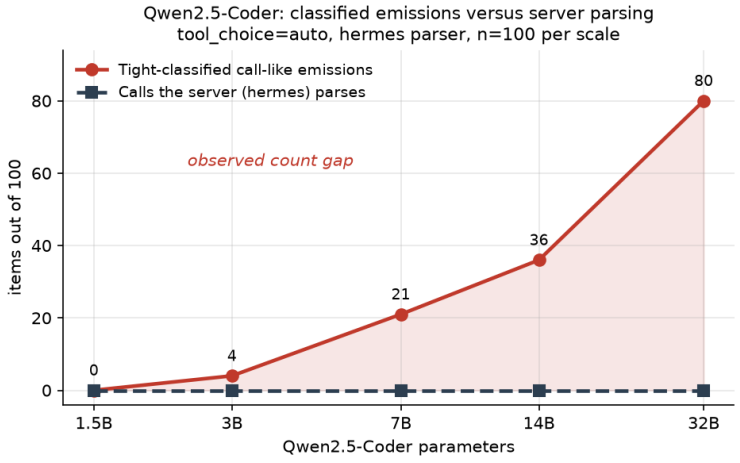


Figure 4: Within-family observation under the mismatched interface. Across the five Qwen2.5-Coder checkpoints, the server parses zero calls while tight-classified emissions rise to 80/100 at 32B (approximately 72 as a sensitivity estimate using tight-stratum precision 0.900, assuming transfer across sizes; Appendix F). Same data as Table 3.

G.6 Main table under unified configuration

Table 27: All arms genuinely at `max_tokens`=2048, one shared 100-item set, `temperature`=0. Mistral has no admissible FC arm: all four configurations produce request errors, so no pass rate and no p-value are computed.

Model	ReAct		FC		b/c	p	FC arm
	turn-1	final	turn-1	final			
Llama-3.1-8B	61	**80**	46	61	27/8	0.0019	official tmpl. + `strict`
Qwen2.5-Coder-7B	57	**74**	53	62	17/5	0.0169	dedicated adapter
Mistral-7B-v0.3	26	33	N/A	N/A	N/A	N/A	—

G.7 The four Llama controls, tabulated

Table 28: Four single-variable controls on Llama-3.1-8B's 23% wrong-tool rate. The first three did not reduce the error; the fourth eliminated it through a schema constraint, without identifying a unique cause of role confusion.

Alternative explanation	Control	Wrong-tool calls / 100	Verdict
schema lacks a parameter description	rich vs terse	23 → 22 (p=1.000)	rejected
ReAct merely adds a reasoning step	FC + Thought scaffold	23 → **59** ($p<0.001$, **worse**)	rejected
server not configured per documentation	+ official chat template	23 → 22 (1/0 discordance, p=1.000)	rejected
schema constraint not enabled	**+ `strict: true` on official template**	**22 → 0 (22/0, p=4.77$\times10^{-7}$)**	**accepted**

G.8 Conditioning on a successful parse, and access vs. use of feedback

Table 29: Conditioning on items where both protocols parsed successfully. These are descriptive post-treatment subsets; neither nominal test clears the 27-test Bonferroni threshold.

Family	Items both parsed	ReAct	FC	b/c	p
Qwen2.5-Coder-7B	83	63	56	11/4	0.118 (n.s.)
Llama-3.1-8B	96	78	60	25/7	0.0021

Table 30: Descriptive repair success among items reaching a second code-bearing, tested step. The subset is selected after treatment; these rates do not localize or causally decompose the protocol gap.

Arm	Reached tested step $\geq$2	Rescued	Repair success given feedback
ReAct	40	17	42.5%
FC + adapter	37	9	24.3%
FC + hermes	**0**	**0**	undefined

Adapters are not a universal switch. The same adapter recovers 84/100 at 7B and **1/100 at 3B**: the 3B checkpoint ignores the `<tools>` few-shot entirely (zero `<tools>` tags in 100 items, 99 direct code) while still solving the task at a comparable rate (final 53). Adapter effectiveness is non-monotonic in scale, reinforcing the per-checkpoint claim of §5.3. The dedicated solution also changes parser, chat template and few-shot simultaneously; we additionally rewrote the template's few-shot examples, which originally invoked `get_weather` — a tool absent from our schema, which would have induced calls to a non-existent tool and inflated both initiation and empty-argument counts. Original and modified hashes are recorded.

G.9 Pressure on a broken interface makes things worse

Two independent controls point the same way. Adding a reasoning scaffold to FC raised role confusion from 23 to **59** (§5.3). Replacing the optional tool instruction with a mandatory one on Qwen2.5-Coder-1.5B moved parser-accepted calls not at all — still 0/100 — while final pass fell from **31 to 15** and unparsable outputs rose from ∼0 to 46/100. Neither intervention touches the interface, and both make the observable outcome worse. **When an agent appears reluctant to use tools, strengthening the instruction is the cheapest thing to try and the most likely to mislead** — it can degrade task performance while leaving the tool-call count at zero, which looks like a model that is both unwilling *and* incapable.

Table 31: Probe replication on the 1.5B model actually being trained, recorded mandatory-prompt override (n=100; exact original file unavailable, Appendix E). Not one output names `run_tests`, so at this scale the missing calls are not a parser mismatch — there was nothing well-formed to censor.

Category	n	%
wrong name, `arguments` carry complete code	**0**	0%
wrong name, `arguments` are the task function's parameters	**36**	69%
wrong name, executable code present elsewhere in the body	16	31%

Table 32: Increasing pressure on the 1.5B model. Stronger instruction and few-shot examples raise the number of recoverable code payloads but never produce a call that names the tool.

	recoverable calls	**naming `run_tests`**	final pass
mandatory prompt (baseline)	52	**0**	15
+ role-disambiguation few-shot	**64**	**0**	**13**

G.10 Formal 7B checkpoint curve and final-lineage audit

All checkpoint evaluations use the same 542-item multi-turn manifest, have zero request failures, and attest the indicated native weight step. The accepted curve uses the original repaired steps 0–90 and resumed evaluation checkpoints 120 and 150; the abandoned original steps 91–116 are excluded. The repeated step-90 evaluation after resumption gave 415/426/11 rather than 415/428/13 and is retained as test–retest evidence, not substituted into the formal curve.

Table 33: Complete formal 7B held-out curve. Each cell is turn-1 pass / final pass / rescued after turn 1, with $n = 542$.

step	broken FC	repaired FC
0	418 / 430 / 12	418 / 430 / 12
30	418 / 430 / 12	418 / 429 / 11
60	420 / 433 / 13	417 / 431 / 14
90	418 / 432 / 14	415 / 428 / 13
120	419 / 433 / 14	415 / 427 / 12
150	418 / 430 / 12	415 / 427 / 12

The policies did update. The accepted logs contain all 150 steps with non-zero actor gradient norms: 0.0120–0.0251 for broken and 0.00641–0.01599 for repaired; mean training rewards span 0.150–0.469 and 0.168–0.435, respectively. A direct checkpoint audit over all 392 LoRA tensors (80,740,352 parameters) finds 45,471,572 values changed from broken step 120 to 150 (56.3%, ℓ_2 delta 0.0787) and 45,296,056 from repaired step 90 to resumed step 150 (56.1%, ℓ_2 delta 0.1222). These comparisons establish updates to saved LoRA weights over the retained intervals; evaluation completion records separately attest the native weight step.

The event join also resolves the funnel's apparent gaps. Broken FC has 19,203 generation events, 8,689 tight emissions and 10 accepted/executed/observed calls. Repaired FC has 33,367 generation events, 17,608 tight emissions, and 16,912 accepted calls in 16,844 accepted generation events.

Forty generations contain multiple calls, producing exactly the 68-call surplus; the configured `max_parallel_calls=1` selects the first returned call and leaves subsequent calls undispatched (neither merged nor queued by this loop). This accounts for the 68-call difference. All 16,844 dispatches join to executions and observations by request ID and assistant turn. Extraction records lack those identifiers; accepted-generation and dispatch counts match per trajectory, rather than through a per-generation join. Tight and accepted are different tests: 16,323 events satisfy both, 1,285 are tight but unaccepted, and 521 are accepted but not tight. With the separately retained full archive (not included in the anonymous package), `analysis/p3_final_evidence.py` verifies all evaluation and event hashes, reconstructs the final lineage, and emits the per-file manifest; the server-side checkpoint audit is reproducible with `analysis/p3_checkpoint_lora_audit.py`.

G.11 Historical ReAct positive control, tabulated

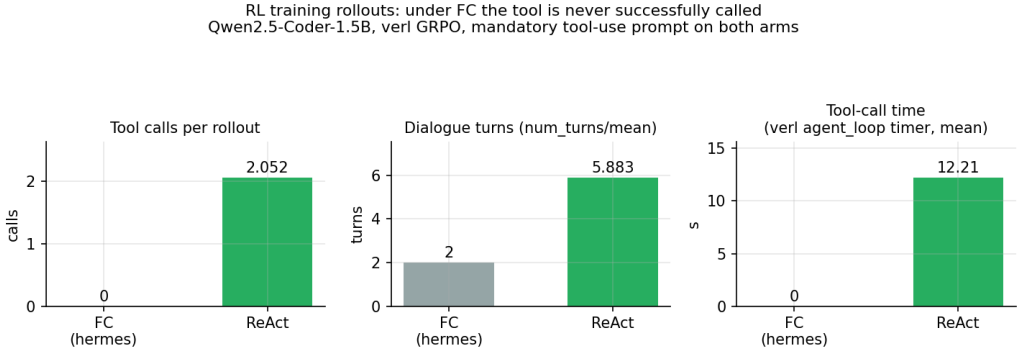


Figure 5: **Historical 1.5B RL diagnostic.** Under function calling the tool is never successfully called; `num_turns` stays pinned at its minimum of 2 and tool time is exactly zero. Prompt strength is matched across arms (both mandatory); step counts are not (10 vs 3). The formal 7B parser control is reported separately in Table 7.

Table 34: Historical ReAct training with a working tool channel, 23,676 executed calls. Rescues by turn $\geq$2 do not move. The three historical 1.5B FC runs, in which **zero** tool calls executed, give rescue sequences $[7, 9, 6, 6, 7, 6, 10, 8, 9, 8, 6]$, $[8, 7, 6, 7, 7, 7, 7, 5, 6, 6, 8]$ and $[7, 5, 7, 5, 7, 8, 9, 9, 8, 9]$. This is supporting evidence only; Table 7 is the single-variable control.

step	turn-1	final	**rescued by turn $\geq$2**	repair channel
0	359	367	8	263
15	340	346	**6**	266
30	340	346	**6**	262
45	341	349	**8**	261
60	341	347	**6**	274

G.12 Sampling variance, and an inverted noise structure

Llama-3.1-8B, `temperature=0.6`, n=100 $\times$ 3 seeds.

Table 35: Sampling variance. FC seed 3 contains one rejected parallel-call request. We report its complete-denominator endpoint, and treat the two clean arms separately; an SD from two clean seeds would be too unstable to summarize variation reliably.

	seed 1	seed 2	seed 3	admissible
ReAct turn-1	53	56	43	3/3
ReAct final	72	72	72	3/3
FC turn-1	44	44	45	—
FC final	62	66	57†	**2 clean / 3 end-to-end**

FC seed 3 contains one request error — the model emitted parallel tool calls and vLLM returned *"This model only supports single tool-calls at once"*. The end-to-end endpoint therefore retains that task as a failure and is 57/100; the prespecified clean-arm analysis excludes the seed and reports [62, 66]. An SD can be computed from two clean seeds, but would be an unstable estimate, so we do not use it to characterize sampling variation. Worst case among clean arms: ReAct's lowest (72) versus FC's highest (66) = **+6**.

Inverted noise structure. ReAct's turn-1 varies widely (43–56, sd 6.8) while its final converges; FC's turn-1 is nearly constant (44–45, sd 0.58) while its final disperses. The mechanism is compensation in the rescue count — 19 / 16 / **29**, largest where turn-1 was worst — and in all three seeds the count of "passed at turn 1 but failed finally" is **0**: the repair loop is monotone. **The three identical finals are a coincidence of totals, not identical item sets**, and sd = 0.00 must not be read as "ReAct is perfectly stable".

G.13 Exploratory observations of stable totals and changing item sets

Two independent observations show the same structure. *Across sampling seeds*: ReAct's final pass is 72 / 72 / 72 at `temperature=0.6`, yet the solved sets have pairwise Jaccard 0.71–0.80 — 15–24 items differ. *Across training steps*: between steps 15 and 30 of the ReAct run, 85 of 542 first-turn completions changed verbatim and 6 items flipped outcome (3 gained, 3 lost), leaving turn-1 and final pass numerically identical (340 / 346 at both checkpoints). The policy is demonstrably moving; the aggregate is not.

These exploratory observations show that equal aggregate scores can conceal different item-level outcomes. Compensation through rescues is consistent with the observed counts, but these few runs do not identify a general stabilising effect of repair or establish a deployment benefit. We therefore report item-level changes alongside totals; both observations come from one model family and one task.

G.14 The training verifier: seams, audit, and planned fixes

The GRPO reward is outcome-only: it extracts the last submitted program, runs the item's own `pytest` suite in the sandbox, and returns the pass ratio (`reward_code.py`). Three reward-hacking routes are closed by construction — a run in which no test executes scores 0 rather than 1, a timeout scores 0, and an item with no test string scores 0. The historical verifier nevertheless had two seams. *(i)* Its summary parser searched `(\d+) passed` over merged stdout/stderr and took the first match, so model stdout could imitate pytest's summary. *(ii)* `skipped` and `xfail` were not counted in the denominator. Both were reachable in principle. We therefore audited: across the five training logs that retain generated code there are **zero** occurrences of a printed `''N passed''`, of `pytest.skip`/`allow_module_level`, or of `sys.exit`/`os._exit`, and `critic/rewards/mean` oscillates in 0.46–0.73 without the step change a discovered exploit produces. The audit's own limit is that historical full rollout text was not retained, so it covers only the fragments those logs carry. Before the formal 7B control we anchored the regex to pytest's summary, included `skipped`/`xfailed`/`xpassed` in the denominator, and retained every rollout in full with a SHA-256 digest. Both P3 arms use this fixed verifier. Neither verifier path is used for the reported held-out pass rates, which require `mode="script"`, a zero exit code and `status=="ok"`.

G.15 Post-hoc common native function-calling evaluation

This additional evaluation uses the existing base and final-lineage step-150 broken/repaired checkpoints, not new training. It was designed after review, outside the training preregistration. All three weights see the same 542 frozen EvalPlus items, original tokenizer/chat template and `run_tests` schema, and the same working repaired verl extraction logic. The BFCL few-shot adapter is not used. Each loaded LoRA has 392 tensors and 80,740,352 parameters; ordered float32 tensor digests match the audited checkpoint exports, with rank/alpha 32 and scaling 1. Base/tokenizer configuration hashes are checked; full base-weight shards were not independently rehashed. Generation uses sequential Transformers with bf16/SDPA, greedy decoding, seed 0, at most four assistant generations of 1,024 new tokens, an 8,192-token context budget and a 1,200-character observation cap.

Context overflow fails explicitly, without silent truncation. The backend is PyTorch 2.13.0, Transformers 5.10.4, PEFT 0.20.0 and safetensors 0.8.0. Anonymized configuration records, including schema, task IDs and source hashes, are included in the package.

Only an accepted, correctly named call with nonempty code triggers interactive execution. The first accepted call per generation is dispatched; additional calls would be ignored, but none occur here. Feedback is a tool-role message and full history is re-rendered on every turn. Passing a tool test alone does not stop the loop; a no-call response or the generation cap ends it. A later explicit final code submission supersedes the last tool code; plain closing prose does not. After the dialogue, final code is scored once independently with no returned feedback. That terminal test is not a tool invocation. Tests use EvalPlus script mode with a 30-second timeout, not the training reward's pytest partial credit. This aligns tool-role feedback, but is not byte-identical replay of AgentLoop or an elimination of task/test distribution shift.

Table 36: Common native FC evaluation, 542 paired tasks per weight. Calls are interactive tool executions; timeouts are terminal scoring timeouts.

Weight	Final pass	Initiating tasks	Calls	Tool rescues	Timeouts
Base	239	46	91	2	4
Broken step 150	248	50	93	1	5
Repaired step 150	248	47	90	1	6

Here a rescue requires a failed first *actual tool test* followed by a passing tool test. An untested first response is not a failure rescued later. Thus these counts are not interchangeable with the earlier automatic-feedback rescue metric. A complementary descriptive count asks whether a failed first tool test is followed by a passing final submission, including a submission not retested interactively: this gives 2/2/1 for base/broken/repaired. Broken MBPP/612 fails three tool tests, then submits passing final code without another tool call. The stricter tool-test rescue count of one therefore does not mean that only one final submission was repaired after feedback. Neither definition shows more rescues for repaired FC. Initiation here means at least one accepted and executed call, not every attempted or call-shaped emission. Accepted calls, executions and returned observations agree (91/93/90); generation counts are 619/622/619. No output hits the generation length cap. Every interactive execution completes with status `ok`; this denotes successful test execution, not necessarily code correctness. Terminal timeouts remain failures in the full 542-item denominator, with no clean-arm filtering. Base timeouts are MBPP 84, 126, 388 and 739; broken additionally times out on 603; repaired additionally on 603 and 765. These observed timeouts are not independently diagnosed as intrinsic nontermination. No missing submissions or other terminal infrastructure statuses are recorded.

The historical programmatic-feedback evaluator records failures on both previously flagged items (HumanEval/32 and MBPP/599); this does not transfer automatically to the new protocol. In native FC, HumanEval/32 fails for all weights, whereas MBPP/599 fails for base but passes for both trained weights. We preserve the frozen 542-item primary analysis. Excluding both historically flagged items gives final success 239/540, 247/540 and 247/540. The trained-arm discordances remain 16/16; the base contrasts lose one trained-only success. This descriptive sensitivity does not reclassify either test as defect-free or diagnose the reason for passing it.

Absolute success also differs substantially across protocols: the trained weights score 430/542 and 427/542 under programmatic feedback versus 248/542 each under native FC. This is not an isolated estimate of the benefit of automatic feedback: the backend, history rendering, code-submission handling and stopping rules also change. We interpret the within-protocol weight contrasts, not the cross-protocol gap as a causal effect. Reduced records cannot independently audit those controller choices; full raw trajectories remain available in the retained research archive.

For repaired minus broken final success, gains/losses are 16/16, difference zero percentage points, conservative paired 95% CI $[-3.47, 3.47]$, exact two-sided McNemar $p = 1$. Relative to base, broken has 20 gains and 11 losses ($+1.66$ points, CI $[-1.74, 5.03]$, $p = 0.150$), and repaired 22 gains and 13 losses ($+1.66$ points, CI $[-1.94, 5.23]$, $p = 0.175$). These intervals use the construction in Appendix I; they cover item-pair uncertainty, not training-seed or inference-retest variability. Equal

totals do not imply equal passing sets or equivalent policies. Neither protocol detects a success benefit of repair in this setting; this does not establish that interface repair cannot improve learning.

The three weight contrasts crossed with final success, initiation and rescue form nine explicitly post-hoc exploratory tests, with Bonferroni-adjusted values supplied alongside nominal values. They are separate from the earlier 27-test reporting audit; none is significant even before adjustment. A single 36-test correction would not make any of the new results significant. The intervals are pointwise, not simultaneous. The deterministic six- and 24-task engineering pilots overlap the full set and are not counted as additional observations or seeds.

The package supplies anonymized configurations, 1,626 item records and `native_fc_results.py`. Running this CPU script with `--outcomes` recomputes every listed contrast and timeout inventory without GPU access. The reduced records do not contain generated code or full conversations; independent reclassification or execution requires the separately retained raw archive. Its `--archive` mode checks raw hashes, matching inputs, loaded adapter identity and execution-to-observation links before reduction.

H LIMITATIONS, IN FULL

In the order a sceptical reader should apply them.

Scope of the evidence. *(1) One task family for the scale and training results.* Our own probe uses decontaminated KodCode items with a single tool taking one `code` argument. The external-validity results are *not* code-only: BFCL's `multi_turn_base` spans file-system, vehicle and trading APIs, and τ-bench `retail` is a customer-service domain whose 789 unparsed turns decompose to 100% envelope-layer with zero parser loss, reproducing the code result (§5.1). What remains code-only is the *scale ladder* and the *RL contamination* result. The former is a real gap — the same ladder on a non-code task would test whether the undercount's growth with checkpoint size is a property of the interface or of code-shaped payloads (§6). The latter we do not expect to close: the reward is a `pytest` pass ratio, so porting it changes environment, reward and task at once. The controlled scale and RL experiments use a single-tool schema and do not isolate tool selection among alternatives. A single-tool schema is an easy parser case, but we treat that only as motivation for broader tests, not as a quantitative lower-bound argument. *(2) Item sampling.* The 50 full-length arms use 100 items taken as `clean[:100]`, **not a random sample**; pairing across arms is exact, so within-study comparisons are sound. Generalisation rests on a replication on a random 300-item sample (seed 20260901): server-parsed remains 0 at every size across 1500 items, and strict emitted counts scale to 0, 4, 30, 40, 81 per 100 against the original 0, 4, 21, 36, 80. **7B moves from 21 to 30**, so the original subset understated that rung; direction and monotonicity are unaffected, but a single 100-item subset is evidently not a stable estimate of an intermediate rung. *(3) The scale result is within one family.* The $0 \to 4 \to 21 \to 36 \to 80$ ladder is Qwen2.5-Coder alone, against one mismatched interface, on one task; we do not claim censoring is generally scale-increasing, and we do not equate parameter count with capability. The Instruct ladder is a control on the *mechanism*, not a second ladder of undercount — its execution layer was initially void (the host lacked `pytest`) and has since been re-run with a working executor, the parse layer reproducing value for value. A second *family*'s ladder remains the test that would promote or refute the scale claim. *(4) The taxonomy is four instances, not an enumeration.* The evidence is deliberately asymmetric: Qwen has a scale ladder, a replay matrix and a repair; Llama a single-variable intervention; **Mistral has no admissible 100-item FC comparison at all**; DeepSeek is a template check with little quantification. *(5) Single-sample headline arms.* Table 20 and the ladders are one sample at `temperature`=0. Variance is estimated only on Llama-3.1-8B, and there only 2 of 3 FC arms are clean. A standard deviation from two seeds is defined but too unstable to summarize sampling variation reliably; the complete-denominator endpoint for the error-bearing seed is still reported (§G.12). *(6) Multiple comparisons.* The explicit 27-test retrospective family in Appendix I uses $0.05/27 \approx 0.00185$. This reporting rule was not preregistered. Surviving it: the official-template `strict: true` intervention on wrong-tool labels ($p = 4.77\times10^{-7}$) and final pass ($p = 7.63\times10^{-5}$), the protocol gap on 300 random items (p=3.1×10^{-6}), and the official-template strict turn-1 contrast ($p = 0.000977$). **Not surviving it**: the Llama strict-to-ReAct contrast ($p = 0.0019$), the both-parsed Llama contrast ($p = 0.0021$), the repair-loop pass gain (p=0.093 at n=100, p=0.0385 at n=300), the Llama protocol contrast at n=300 (p=0.0352), and the residual-gap estimates (p=0.118, 0.125). We rest no claim on the latter group,

and report the n=300 values precisely so a reader can see the direction is stable where significance is not claimed. *(7) Model scale stops at 32B*, on a single A800; no frontier models.

Threats to the causal reading. *(8) The clean repaired-FC control has one seed and one budget.* The completed 7B experiment holds model, LoRA configuration, data, prompt, decoding, verifier, seed and 150-step budget fixed, changing only the parser registered in verl. It therefore identifies the interface effect on the sampled experience distribution in this setting. Its null held-out result does not establish that a working channel is generally insufficient, and its LoRA training is not strictly comparable to the historical 1.5B full-parameter runs. *(9) Conditioning on a successful parse is post-treatment conditioning.* Whether an item parses is determined by the interface under test, so the both-parsed subset is not a random subsample and its contrast is not a clean causal estimate. On Qwen the residual is $+8.4$ pp, 95% CI $[-0.5, +17.4]$, p=0.118: **we do not detect a residual difference, which is not the same as establishing there is none**. *(10) Near-zero observed frequency is not zero probability.* The broken 7B arm executes 10 calls rather than none; its instrumentation describes a sampled distribution, not the support of the policy. We therefore report a nearly closed channel (10 executions alongside 8,689 tight-classified generation events), not a mathematically unreachable action. *(11) Attempts and discards are separated only in the new run.* The formal 7B arms retain complete rollout text with hashes, parser verdicts, execution events and returned observations, so 8,689 regex-positive emissions and 10 accepted actions are separately recorded; the classifier is not a semantic-validity oracle and its link to dispatch is a trajectory-level count check. The historical 1.5B run did not retain full text. Its same-prompt replication finds no correctly named call, but that cannot retroactively identify what every historical rollout emitted. *(12) The null learning observation is setting-specific.* Parser repair restores 16,844 executed and observed calls over 150 steps with 12 held-out rescues at both endpoints (intermediate checkpoints vary). This is the requested single-variable control, but it is one seed, 7B, LoRA, one code task and one training horizon. The earlier ReAct arm (23,676 calls, flat rescues) is directionally consistent but remains confounded by its protocol change.

The shared evaluation deliberately avoids both training parsers, but changes structured tool-role feedback to user-message feedback. It measures transfer to this programmatic-feedback protocol, not native FC behavior. Formatting shift, limited adaptation budget/capacity and multi-turn credit assignment are plausible explanations for absent transfer; these data do not separate them. The evaluator extracts the last decodable `<tool_call>` code argument when present, otherwise the last code fence or the raw text. This is not a general decoder for every tool syntax. The post-hoc common native FC evaluation (Appendix G.15) additionally uses voluntary calls and tool-role feedback: repaired and broken both pass 248/542, with 16 discordances each way. This addresses the missing initiation measurement, not all distribution shift: held-out tasks use script-mode tests, the controller re-renders history, and only one training seed is available. It is not a byte-identical training rollout or an equivalence test. The native FC tool-test rescue definition requires an actual failed tool test followed by a passing tool test; a final-submission alternative is reported separately. Neither may be pooled with programmatic-feedback rescues. Absolute scores differ across evaluation protocols, so their cross-protocol gap must not be attributed solely to whether testing is automatically initiated.

(13) Human validation covers one criterion on one sample. The 98-item stratified sample has 97 complete paired annotator records; these give inter-annotator κ = 0.967 on the 62 items where both read identical bytes, and classifier-vs-gold-standard κ = 0.871 after third-party adjudication (Appendix F). Two residual limits: the sample is stratified by classifier verdict, so the unweighted sample metrics are not population precision or recall; and stored raw output is capped at 4000 characters, so a call emitted past that point is invisible to classifier and annotators alike, creating false negatives. Because the classifier also has false positives, the net bias of the raw emitted-call count is not identified. The *first* round's apparent unreliability (κ=0.713) was confounded by 2400-character truncation in the annotation pack and by selective adjudication; the identical-byte analysis removes the presentation confound but does not establish that every disagreement was an instrument error. *(14) The historical training verifier had two exploitable seams.* Its summary parser used the first unanchored `(\d+) passed` match and omitted `skipped`/`xfail` from the denominator. We found no exploit signature in the five historical logs that retain generated code, but those logs do not contain every full rollout. For the formal 7B control we anchored the regex to pytest's summary, included skipped, xfailed and xpassed outcomes in the denominator, and retained every rollout in full with a hash. Both arms use the identical fixed verifier, so it cannot explain their contrast. Neither verifier version is used for the separately reported held-out pass rates (§G.14).

Environment dependence and provenance. *(15) Serving-stack version dependence.* Serving and training measurements use vLLM 0.27.1 and verl 0.9.0, so parser behaviour, template defaults and the semantics of `strict: true` are properties of those versions. We pin versions, parser and template hashes, and model revisions for exactly this reason, and we have not replicated on a second serving stack (e.g. SGLang), so we cannot say which findings are vLLM-specific. The post-hoc native FC evaluation uses Transformers 5.10.4/PEFT 0.20.0 rather than a second serving-stack replication; its controller and test protocol differ from training and the earlier evaluator. *(16) Repository-revision dependence.* Chat templates ship inside `tokenizer_config.json` and can be changed by the publisher without a model change; the Qwen2.5-Coder false positive is precisely an artefact of an inherited template. *(17) Instrumentation errata, disclosed rather than absorbed.* Seven ReAct arms recorded `max_tokens=2048` while running at 1024 (an asymmetry that *favoured* FC; re-running changed final pass by $\leq$1 item); Llama's initiation rate required manual correction from 97 to 74 after tool-name validation was added; five arms are error-bearing and admissible only for the error-rate census, and one previously reported p=0.648 computed on such an arm is withdrawn. All are enumerated in Appendix A. None is silently absorbed into a reported number.

What would change our conclusions (restated)

We name these so the claims are falsifiable rather than merely hedged. *Replications or a longer repaired-FC budget.* The requested broken-versus-repaired single-variable control is now complete: parser repair restores 16,844 tool interactions and does not increase held-out rescues. A consistent rescue gain over additional seeds or a longer preregistered horizon would narrow the scope of this 150-step, single-seed observation; continued nulls would strengthen it within, but not beyond, the tested model, task and optimiser setting. *A second family's scale ladder*, with a working test executor: if the undercount does not grow with checkpoint size in another family, §5.2's scale claim is a Qwen-plus-`hermes` fact rather than a scaling phenomenon. *The same ladder on a non-code task.* The five Qwen2.5-Coder checkpoints run against τ-bench `retail` under the documented configuration — models and interface fixed, only the domain varied. If the emitted column still rises with size while the parsed column stays at zero, the scale result is a property of the interface; if it does not, it is a property of code-shaped payloads. This needs a rescue arm at the small sizes for the reason Granite did: a checkpoint that simply never attempts a call produces an uninformative zero, not a censored one. *The same five-layer measurement on further standard benchmarks.* Done for BFCL v4 and for τ-bench `retail` (§5.1), so the measurement-validity concern is *not* specific to our harness, and it now covers one interactive suite with a simulated user and stateful environment. What the τ-bench arm does *not* establish is an outcome effect: 7 versus 10 tasks solved on three discordant pairs is far from significant, and its user simulator is a local Llama-3.1-8B rather than `gpt-4o`, so its absolute scores are not comparable to the published leaderboard. Suites whose tool schemas are much larger, or whose scoring depends on argument-level matching, remain untested.

I Statistical estimands and retrospective test family

I.1 Finite-sample interval for the formal RL endpoint

Let p_+ denote the probability that an evaluation item passes only under repaired FC and p_- the probability that it passes only under broken FC. The paired risk difference is $\Delta = p_+ - p_-$. At step 150, the counts are $(n_+, n_-, n) = (0, 3, 542)$. For each discordance probability we invert the binomial distribution to obtain a two-sided 97.5% Clopper–Pearson interval, denoted $[L_+, U_+]$ and $[L_-, U_-]$. The union bound gives at least 95% coverage for $[L_+ - U_-,\ U_+ - L_-]$ under independent evaluation-item pairs. This conservative construction does not rely on a normal approximation with three discordant pairs. The point difference is -0.55 percentage points and the interval is $[-1.79, 0.72]$ percentage points. For repaired step 0 versus 150, $(1, 4, 542)$ gives $[-2.06, 1.01]$ percentage points. These are unadjusted endpoint intervals, not simultaneous intervals for the whole test family. They quantify item-pair uncertainty under an independence assumption, not training-seed, inference-retest, benchmark-selection or task-cluster uncertainty.

I.2 EXPLICIT REPORTING FAMILY

This is a retrospective reporting audit, not a preregistered multiplicity policy. The family includes all 15 metrics in the five audited Llama control contrasts, even when only one metric is highlighted, plus the 12 distinct comparisons below. Repeated appearances of the same contrast count once. Nominal p values are exact two-sided McNemar tests; adjusted $p = \min(1, 27p)$, with threshold $0.05/27 = 0.00185185$. Historical 1.5B motivating runs and abandoned/invalid runs are not confirmatory evidence and are outside this family; their original disclosures remain in the historical audit. The two both-parsed contrasts are retained in the family but remain post-treatment descriptive comparisons, not causal effects. This explicit family replaces the earlier approximate count of twelve; in particular, the Llama strict-to-ReAct and both-parsed contrasts no longer clear the corrected threshold.

Table 37: Retrospective test inventory. G/L are later-arm gains/losses in the named binary metric; for wrong-tool metrics a gain means more wrong-tool calls, not an improvement. A star marks $p \leq 0.05/27$.

ID	Contrast and metric	n	G/L	nominal p
T01	Llama terse to rich: wrong_tool	100	5/6	1.00
T02	Llama terse to rich: turn1_pass	100	4/6	0.754
T03	Llama terse to rich: final_pass	100	10/10	1.00
T04	Llama terse to thought: wrong_tool	100	37/1	2.84e-10*
T05	Llama terse to thought: turn1_pass	100	1/23	0.00000298*
T06	Llama terse to thought: final_pass	100	4/29	0.0000109*
T07	Llama terse to official: wrong_tool	100	0/1	1.00
T08	Llama terse to official: turn1_pass	100	0/1	1.00
T09	Llama terse to official: final_pass	100	1/6	0.125
T10	Llama terse to strict: wrong_tool	100	0/23	2.38e-7*
T11	Llama terse to strict: turn1_pass	100	13/1	0.00183*
T12	Llama terse to strict: final_pass	100	15/3	0.00754
T13	Llama official to strict: wrong_tool	100	0/22	4.77e-7*
T14	Llama official to strict: turn1_pass	100	14/1	0.000977*
T15	Llama official to strict: final_pass	100	18/1	0.0000763*
T16	Llama strict to ReAct final	100	27/8	0.00188
T17	Llama terse to ReAct final	100	38/7	0.00000312*
T18	Qwen Hermes to adapter final	100	16/7	0.0931
T19	Qwen adapter to ReAct final	100	17/5	0.0169
T20	Qwen Hermes to adapter final (random)	300	43/25	0.0385
T21	Qwen adapter to ReAct final (random)	300	59/18	0.00000306*
T22	Llama strict to ReAct final (random)	300	51/31	0.0352
T23	Qwen both-parsed FC to ReAct final	83	11/4	0.118
T24	Llama both-parsed FC to ReAct final	96	25/7	0.00210
T25	tau documented to repaired success	115	3/0	0.250
T26	P3 broken to repaired final at step150	542	0/3	0.250
T27	P3 repaired step0 to step150 final	542	1/4	0.375

The accompanying `submission_statistics.py` recomputes the 15 Llama tests from item-level records and the remaining tests from the explicit paired counts, with their sources identified in `submission_statistics.json`. Thus nominal and adjusted values can be reproduced without GPUs. The latter count-based recomputation is not a new item-level audit of every historical comparison.

The later post-hoc native FC evaluation adds a separately identified nine-test exploratory family (Appendix G.15): three paired weight contrasts crossed with final success, tool initiation and rescue. Its per-item inputs, nominal and Bonferroni-adjusted values are supplied separately. No new contrast is nominally significant, and none becomes significant if all 36 tests are pooled. The historical 27-test audit is preserved, not retroactively called preregistered.