# PROOF-Gen: From Optimized Data to Better Distillation

**Anh Ta** **Junjie Zhu** **Shahin Shayandeh**
Apple
{atta, jason.zhu, shn}@apple.com



## Abstract

Supervised fine-tuning on teacher-generated trajectories is the standard first stage for distilling tool-calling capabilities into deployable models. Post-training pipelines that drive shipped tool-calling agents re-run this stage on a daily or weekly cadence, paying the frontier-teacher cost each cycle, yet the mechanism is generate-and-filter (keep the teacher's passing trajectories, discard the rest) and each cycle leaves behind the same hard scenarios because failures supply no signal. On $\tau$2-bench, 57% of teacher trials fail, two-thirds of them *near-misses* (most tool calls correct, undone by one decisive error).

We introduce PROOF-Gen (Per-scenario Reflective Optimization to Overcome Failed Generation), which recovers golden trajectories from these failures via per-scenario prompt optimization. For each failed task, a reflector analyzes the execution trace and evaluation feedback, then writes corrective guidance that steers the teacher to a passing trajectory. The guidance is stripped before training, so the student learns from clean demonstrations with no task-specific scaffold.

On $\tau$2-bench, per-scenario optimization recovers 93% of failed scenarios. Fine-tuned on the combined data, Qwen3-4B-Instruct-2507 improves from Pass^1=0.132 to 0.529 and Gemma 4 E4B-it gains +7.2pp on BFCL v4 multi-turn. In a deployed pipeline, the method lifts trajectory quality by +6.3pp goal completion and transfers to a deployed on-device model (+1.5pp goal completion; +1.7 to +5.0pp across response-quality metrics), with positive transfer in every locale (non-English average +1.48pp).

## 1 Introduction

Distillation via supervised fine-tuning (SFT) is the dominant first stage for transferring capabilities from frontier models to the smaller, deployable models that drive tool-calling agents in production (Agarwal et al., 2024; Li et al., 2025; Luo et al., 2026). Post-training cycles run on a daily or weekly cadence to absorb new scenarios, tools, and policies; every cycle pays the same frontier-teacher cost, and every cycle leaves behind the same hard scenarios, because the standard generate-and-filter mechanism (run the teacher, score with a verifier, keep the passing trajectories) extracts no signal from failures. Recent advances in the training algorithm (on-policy sampling, alternative divergences, partial supervision; Agarwal et al. 2024; Luo et al. 2026; Amani et al. 2026), data selection (Li et al., 2025), and trajectory structure (Jiang et al., 2026) all rest on a shared assumption: the teacher can produce correct demonstrations for the tasks that matter.

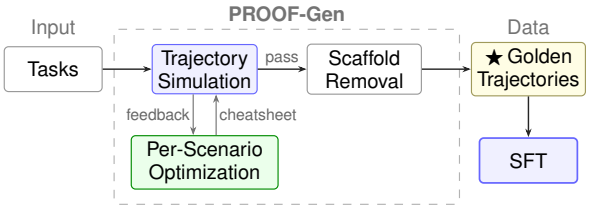


Figure 1: Training pipeline. For each task the teacher originally failed, a per-scenario optimization loop (green) iteratively refines a cheatsheet from the trajectory's feedback until the teacher produces a passing ("golden") trajectory. The cheatsheet is stripped at scaffold removal, so the student trains only on what the trajectory demonstrates; the fine-tuning procedure is otherwise unchanged.

But Chu et al. (2025) show that SFT tends to memorize its training distribution rather than generalize beyond it. If the hardest scenarios are systematically absent from that distribution, no training algorithm or selection strategy can compensate; the ceiling on distillation quality is set upstream, at data generation time.

In settings with execution-based verification, the standard pipeline (Chen et al., 2023; Liu et al., 2024) is generate-and-filter: execute the teacher,

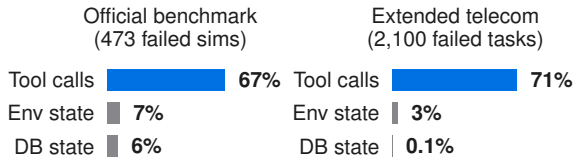


Figure 2: The accuracy asymmetry in failed GPT-4o trajectories on $\tau$2-bench (278 tasks, 3 trials each; 473 of 834 trials fail). Most failures execute over half of tool calls correctly (high tool-call accuracy), yet a single missed action cascades into an incorrect end state (environment and database failure). Because tool-call accuracy is already high in failures, turn-level or partial-credit signals saturate without lifting end-to-end completion; what is missing is a complete trajectory through the decisive step. We confirm this in Section 4.

score with a verifier, and keep passing trajectories (rejection sampling; Yuan et al. 2023; STaR; Zelikman et al. 2022; ReST; Gulcehre et al. 2023). The filter is binary; a trajectory that completed all but one of its required actions is discarded identically to one that failed immediately, so hard cases produce no training signal. On $\tau$2-bench, 67% of failed GPT-4o trajectories execute over half of the required tool calls correctly, yet a single missed action cascades into full failure and the entire trajectory is discarded (Figure 2).

This asymmetry also explains why turn-level alternatives to binary filtering are insufficient. The natural response to discarding a near-complete trajectory is to award partial credit for the actions it got right. But Figure 2 shows tool-call accuracy is already high in failures; rewarding correct intermediate actions reinforces behavior the teacher already exhibits. What the student never observes is a complete trajectory that navigates the single decision point where the teacher fails. We confirm this empirically in Section 4: training on the partial-credit baseline (the filtered-only condition) lifts tool-call accuracy by 25pp but does not improve end-to-end task completion in any model–benchmark pair.

Likewise, sampling more, raising temperature, or running k-shot rejection recovers stochastic failures but not strategic ones: a single missed action at a state-dependent decision point survives any number of resamples from the same teacher.

We introduce Per-scenario Reflective Optimization to Overcome Failed Generation (PROOF-Gen), which brings prompt optimization into this pipeline but inverts its usual goal: instead of finding one prompt that generalizes across tasks, we optimize each scenario independently (Figure 1). Using GEPA (Agrawal et al., 2026), an iterative prompt optimizer, we analyze each failed execution trace and evaluation feedback, then add corrective instructions to a per-scenario *cheatsheet* appended to the teacher's system prompt. The teacher re-executes the task from scratch, and the process repeats until all evaluators pass. The cheatsheet is stripped before training (Figure 1); the student learns from what the trajectory demonstrates, not from the scaffold that produced it. The method applies to any domain with an executor and a verifier.

This design also yields *capability–voice decoupling*: because the executor is left unchanged, recovered trajectories inherit a stronger reflector's problem-solving without its interaction style. The resulting distillation is *voice-preserving*, letting a team upgrade task coverage without disrupting an established product voice, latency profile, or tone (Section 4).

We organize our investigation around three questions: whether per-scenario optimization reliably recovers high-quality trajectories from teacher failures (**RQ1**); whether students trained on recovered trajectories outperform those trained on filtered-only data, the partial-credit baseline (**RQ2**); and whether the approach holds up beyond benchmarks under deployment-scale evaluation across tens of locales (**RQ3**). We address each in turn in Section 4.

We conduct our primary study on $\tau$2-bench (Barres et al., 2025), a stochastic multi-turn tool-calling benchmark, with additional experiments on BFCL v4 multi-turn (Patil et al., 2025). The teacher is GPT-4o, whose low pass rate on the official benchmark[1] leaves substantial room to validate recovery. Two small instruction-tuned models fine-tuned on recovered data improve on both benchmarks, and the pipeline has been adopted within a production tool-calling agent post-training system; we report all three settings in Section 4.

## 2 Method

PROOF-Gen adds one step to the standard generate-and-filter pipeline: it recovers the trajectories that filtering discards. As usual, the teacher generates one trajectory per task and we

[1] https://llm-stats.com/models/compare/gpt-4o-2024-08-06; GPT-4o scores 23.5% on telecom, 45.5% airline, 63.4% retail.

keep the passing ones (the *filtered* set). For each *failed* task, we then optimize a disposable, per-scenario prompt until the teacher succeeds, strip that prompt, and add the recovered trajectory to the training set (Figure 1). The student is fine-tuned on the union of filtered and recovered trajectories.

**Per-scenario recovery.** A reflector model reads the failed execution trace and the verifier's feedback (which dimensions failed, and on which assertions), then writes a *cheatsheet*: a block of natural-language guidance appended to the teacher's system prompt for that one task. The teacher re-executes from scratch; if it now passes every evaluator the trajectory is kept, otherwise the reflector revises the cheatsheet and the teacher tries again, up to $K$=10 iterations (each iteration sees the latest trace and feedback, and the cheatsheet accumulates the diagnoses so far). We use GEPA (Agrawal et al., 2026) as the optimizer, though any prompt optimizer would serve. This inverts the usual goal of prompt optimization: rather than searching for one prompt that generalizes across inputs, we overfit a prompt to a single scenario and discard it. The cheatsheet is the means; the passing trajectory is the product. Its guidance is procedural (e.g., the order of diagnostic tool calls), not task-specific values (Figure 11). Scenarios still unsolved after $K$ iterations are discarded; recovery rates are reported in Section 4.

**Scaffold removal.** The cheatsheet is appended with a fixed delimiter and deleted before training, so recovered trajectories are indistinguishable from filtered ones and the student never sees per-scenario scaffolding. Any gain after fine-tuning therefore reflects what the trajectories demonstrate, not the prompt that produced them; we confirm the cheatsheets supply guidance rather than answers in Section B.1.

**Configurable objective.** Because the reflector hill-climbs on whatever the verifier reports, the same loop can target any combination of evaluable quality dimensions (e.g., groundedness, brevity, formatting), not only task completion; we use this in the production setting (Section 4).

## 3 Experimental Setup

We evaluate on two public benchmarks whose structure mirrors production tool-calling post-training systems: scenarios define user requests, execution-based evaluators serve as quality checks, and a teacher model generates candidate trajectories. The key differences in production are scale (thousands of scenarios with longer trajectories), multi-objective optimization (multiple evaluators must pass simultaneously), and continuous training (daily data ingestion and retraining); we address these in Section 4.

| | $\tau$2-bench | BFCL v4 |
|---|---|---|
| Eval tasks | 114 | 240 |
| User simulation | Stochastic | Deterministic |
| Primary metric | Pass^1 | Task accuracy |
| Secondary metric | Partial reward | Partial reward |

Table 1: Benchmark comparison. $\tau$2-bench uses an LLM user simulator; BFCL uses pre-scripted user turns.

**Benchmarks.** $\tau$2-bench (Barres et al., 2025) simulates multi-turn customer service interactions with an LLM user simulator, making evaluation stochastic. We use the telecom domain (2,285 tasks, ∼50 tools) for primary experiments: it has the largest task space and the lowest pass rate under generate-and-filter (7.3%), maximizing the number of failures available for recovery (RQ1). The split is at the task level: 114 held-out base tasks form the evaluation set and the 2,171 non-base tasks are used for training, with no task instance shared. Because $\tau$2-bench tasks are parameterized from templates, train and eval may instantiate shared templates with different parameters; we do not enforce template-level disjointness (Appendix J). BFCL v4 multi-turn (Patil et al., 2025) evaluates function-calling across 162 functions with deterministic (pre-scripted) user turns, spanning four categories that test distinct failure modes (Appendix L). Each category contains 200 tasks (800 total); we apply a per-category 70/30 split, giving 560 training and 240 held-out test tasks. We include BFCL to confirm that downstream gains (RQ2) generalize beyond a single stochastic benchmark. Table 1 summarizes both.

**Models.** We fine-tune two small instruction-tuned models at comparable effective scale: Gemma 4 E4B-it (Google DeepMind, 2026) (4.5B effective parameters) and Qwen3-4B-Instruct-2507 (Yang et al., 2025) (4B parameters). They differ in architecture and tool-call format, so improvements on both suggest the approach is not model-specific (Appendix I). The teacher is GPT-

| | $\tau$2-bench telecom | BFCL v4 multi-turn |
|---|---|---|
| *Generate-and-filter (teacher)* | | |
| Training pool (tasks) | 2,171 | 560 |
| Filtered (passed) | 158 | 296 |
| Failed | 2,013 | 264 |
| Teacher pass rate | 7.3% | 52.9% |
| *Per-scenario recovery* | | |
| Failures sent to optimizer | 300 | 264 |
| Recovered | 279 | 89 |
| Recovery rate | 93.0% | 33.7% |
| *Combined training set (trajectories)* | | |
| Total (filtered + recovered) | 437 | 385 |
| From recovery | 64% | 23% |

Table 2: Data-generation pipeline for both benchmarks. The teacher is GPT-4o (temperature 0, one trajectory per task). Telecom recovery runs on a 300-failure sample of the 2,013 failures; all 264 BFCL failures are attempted. Recovery rate = recovered / failures sent to optimizer. The combined set merges filtered and recovered trajectories; "from recovery" is the recovered fraction of that set.

4o;[2] the primary reflector is GPT-5.1 for $\tau$2-bench and GPT-5.4 for BFCL (both at high reasoning effort); each is the strongest reflector available when that benchmark was run.

**Data generation.** GPT-4o generates one trajectory per task at temperature 0. Table 2 summarizes the resulting pipeline for both benchmarks. The teacher pass rate differs sharply (7.3% on telecom vs. 52.9% on BFCL), and the recovery rate scales inversely with it: per-scenario optimization recovers 93.0% of attempted telecom failures but 33.7% on BFCL, where the teacher already solves the easier tasks and only the hardest failures remain. Telecom yields 2,013 failures, so we attempt recovery on a fixed random sample of 300, keeping teacher-passing trajectories a meaningful share of the combined set (36%); BFCL yields only 264, small enough to attempt in full. Full per-category counts and train/test splits are in Appendix J.

**Training.** QLoRA (Dettmers et al., 2023) for 3 epochs on both models. Hyperparameters are in Appendix E.

**Evaluation.** End-to-end task success is the primary metric. $\tau$2-bench evaluates along three programmatic dimensions (action, environment, and database; defined in Appendix A.1), with telecom using environment assertions as the pass/fail criterion. We run 3 trials at temperature 0; Passˆ1 is the fraction of tasks passing at least once. At temperature 0 the agent decodes deterministically; the 3 trials average over the LLM user simulator and vLLM serving non-determinism. BFCL reports task accuracy and turn-level partial reward. Sampling robustness is tested at temperature 1 on flip tasks (Zhai et al., 2026).

[2] `gpt-4o-2024-08-06` throughout.

## 4 Results

We organize our results around three research questions:
**RQ1**: Does PROOF-Gen reliably recover high-quality trajectories from teacher failures?
**RQ2**: Do models trained on recovered trajectories outperform those trained on filtered-only data?
**RQ3**: Does PROOF-Gen scale to production systems that continuously do SFT retraining on large-scale datasets with multiple objectives?

**RQ1: Recovery effectiveness.** At production scale, per-scenario optimization recovers 93% of sampled telecom failures (279 of 300; GPT-4o executor, GPT-5.1 reflector; Table 3); 264 (88%) are driven by the optimized cheatsheet and 15 by stochastic replay of the user simulator. We report the inclusive 93% as the headline rate and the cheatsheet-attributable 88% where the mechanism is the focus. Recovery is driven by reflector capability: on a cross-domain ablation set of 82 failures spanning all three $\tau$2-bench domains, it ranges from 24.4% (GPT-4o self-reflection) to 97.6% (GPT-5.1; Figure 3). Convergence is fast: 58.7% of cheatsheet-driven recoveries resolve on the first iteration and 84.1% by the third (Figure 6). Even GPT-5.1 at maximum reasoning budget benefits from one reflection step (83.3% $\rightarrow$ 98.3% on the 300 telecom tasks; Table 3). Recovered trajectories constitute 64% of the combined $\tau$2-bench training set; on BFCL, 23%. The cheatsheet changes what the teacher does, not how it communicates (Section 4). Integrity audits confirm no factual leakage (Section B.1).
*Details: Appendix A (recovery rates, convergence, frontier comparison); Appendix B (data quality audit, response complexity).*

**RQ2: Downstream impact.** The filtered-only condition is the partial-credit baseline: it trains on exactly the turn-level signal that is already high in failures (Figure 2). As predicted, filtered-only training improves partial metrics (action accuracy +25pp) but does not improve task com-

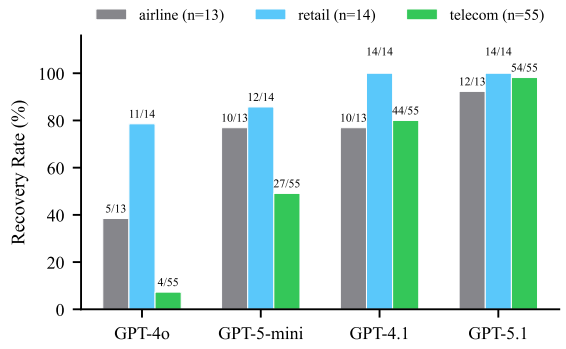


Figure 3: Reflector capability dominates recovery rate (range 24.4%–97.6%). Per-scenario recovery by reflection model on a cross-domain ablation set of 82 failed $\tau$2-bench scenarios (13 airline, 14 retail, 55 telecom), holding the executor (GPT-4o) and optimizer fixed. GPT-5.1 recovers 80/82 (97.6%) versus 20/82 (24.4%) for GPT-4o self-reflection; the largest gap is on telecom (54/55 vs. 4/55). This ablation isolates the effect of reflector capability and is not the source of the production-scale headline rate (93%, 279/300 telecom).

pletion in any of the four model–benchmark pairs (Figure 4); raising intermediate-action accuracy is not sufficient when the missing signal is a complete trajectory through the decision point. Combined training produces the only consistent gains: Qwen3-4B-Instruct-2507 (Yang et al., 2025) improves from Pass^1=0.132 to 0.529 on $\tau$2-bench; Gemma 4 E4B-it (Google DeepMind, 2026) gains +7.2pp on BFCL. The recovered trajectories cover tasks the teacher failed without the cheatsheet (GPT-4o scores 0.205 on these tasks); produced by GPT-4o under a stronger reflector, they let the student inherit problem-solving the base teacher does not exhibit on its own. Gains persist under stochastic sampling at temperature 1: on flip tasks (post-hoc selected where fine-tuning helped at temp=0; $n$=46 Qwen, $n$=21 Gemma), the combined model averages 54.1% (Qwen) and 39.0% (Gemma) per-trial success, versus near-zero baselines (Figure 7).

**Capability vs. voice: why the executor matters.** Per-scenario optimization separates two factors that direct distillation conflates: the reflector supplies task-solving *capability*, while the executor supplies interaction *voice*. On 259 matched telecom tasks, the cheatsheet changes which tools GPT-4o calls but not how it communicates: GPT-4o keeps its terse, incremental turn structure (one diagnostic at a time), whereas GPT-5.1 front-loads a single numbered multi-step protocol 3.8× longer at identical tool-call counts (Appendix B.3, H). PROOF-Gen therefore yields *voice-preserving distillation*: a team can adopt a stronger reflector's problem-solving without inheriting its voice or latency profile.

**RQ3: Production scaling.** To test whether PROOF-Gen transfers beyond the controlled $\tau$2-bench and BFCL benchmarks, we deployed it in a production tool-calling agent post-training system. Evaluation is end-to-end: thousands of held-out scenarios are run in simulation and scored by deterministic verifiers together with LLM judges, across goal completion and five response-quality metrics (entity formatting, brevity, groundedness, tool-calling accuracy, style and tone). Absolute values are withheld per deployment policy, so we report percentage-point (pp) gains over the respective baselines; as the production evaluation is a single large-scale run, we treat consistency across metrics and locales as the robustness signal. Per-scenario optimization improved generated training-trajectory quality by +6.3pp in goal completion and +2.5 to +8.0pp across the five response-quality metrics. These gains transferred to the downstream on-device model, evaluated without any cheatsheet augmentation: +1.5pp goal completion and +1.7 to +5.0pp across the same metrics, with no regression on any dimension. The effect holds across tens of locales (positive transfer in every evaluated locale; non-English average +1.48pp goal completion, range +0.15 to +3.38pp), indicating the corrective knowledge is largely language-agnostic. We present this as a deployment case study corroborating the controlled, reproducible benchmark results.

**Efficiency and orchestration.** Per-scenario optimization is fully parallel: scenarios are optimized independently with no shared state, so the recovery stage fans out across workers. It is also cheap in frontier calls: the expensive reflector only diagnoses and revises the cheatsheet while the cheaper executor handles the rollout, far fewer frontier calls than using a frontier model as the executor at every turn (Appendix A.3). Implementation is standard: an off-the-shelf prompt optimizer (GEPA via DSPy) wrapped in any job scheduler (e.g., Ray, Airflow).

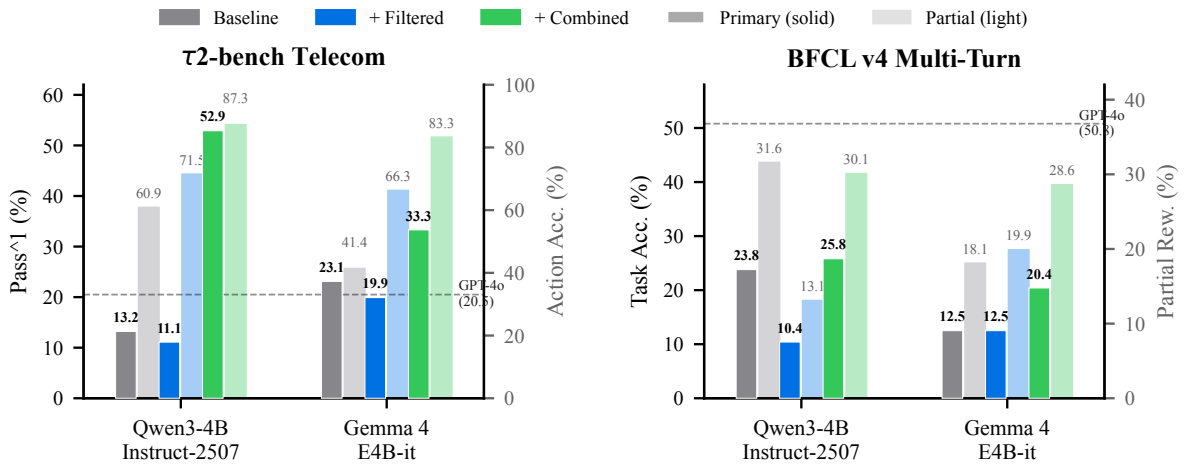


Figure 4: Post-training results on $\tau$2-bench telecom (left) and BFCL v4 multi-turn (right). Solid bars show the primary metric (Pass^1, task accuracy; left axis); hatched bars show partial rewards (action accuracy, turn-level partial reward; right axis). Dashed lines mark the GPT-4o teacher. Filtered-only training improves partial rewards but does not improve end-to-end task completion in any of the four model–benchmark pairs; combined training produces the only consistent gains.

## 5 Related Work

PROOF-Gen addresses the coverage gap that generate-and-filter leaves in every training cycle; we situate it against three lines of work.

**SFT for tool-calling agents.** Training small models to use tools via SFT on teacher-generated trajectories is well established (ToolLLM (Qin et al., 2024), FireAct (Chen et al., 2023), APIGen (Liu et al., 2024), xLAM (Zhang et al., 2025), and Magnet (Yin et al., 2025)). All treat data generation as uniform sampling from the teacher, keeping whatever passes and discarding the rest; our contribution is recovering the failures.

**Trajectory recovery and self-improvement.** Methods that reuse failures (STaR (Zelikman et al., 2022), ReST$^{\text{EM}}$ (Singh et al., 2024), V-STaR (Hosseini et al., 2024), ETO (Song et al., 2024), Agent-R (Yuan et al., 2025), and Reflexion (Shinn et al., 2023)) share a structural assumption: a successful trajectory already exists for each failure, supplied by an expert, a stronger model, or a passing rollout to pair against (offline preference learning in V-STaR and ETO). PROOF-Gen targets the upstream problem: it *creates* a successful trajectory where no expert solves the task, modifying only the teacher's prompt (no weight updates, tree expansion, or failure examples). The two are complementary: recovered trajectories can feed any preference method that needs paired data.

**Prompt optimization as a data-generation instrument.** Automatic prompt optimization typically searches for one prompt that generalizes across inputs at inference time (Pryzant et al., 2023; Yang et al., 2024; Opsahl-Ong et al., 2024; Khattab et al., 2024; Xu et al., 2024). ACE (Zhang et al., 2026) is the closest analog, accumulating task-specific "playbooks" from execution feedback like our cheatsheets, but deploys them as persistent inference-time context. We instead treat the prompt as a disposable instrument for exhausting a single scenario's solution space and discard it before training, retaining only the passing trajectory; to our knowledge, using automatic prompt optimization explicitly as a data-generation engine has not been previously studied.

## 6 Conclusion

Generate-and-filter distillation discards the teacher's hardest failures, leaving a coverage gap no downstream training algorithm can close. PROOF-Gen treats them as recoverable: per-scenario prompt optimization steers the teacher to a passing trajectory, and the scaffold is stripped so the student learns from clean demonstrations. It recovers 93% of sampled telecom failures, lifts downstream task completion for two small instruction-tuned students on $\tau$2-bench and BFCL (the only condition to do so in all four pairs), and improves a production agent post-training system with positive transfer in every locale.

## Limitations

**Reflector generalization.** We characterize the recovery-vs-capability relationship across four models from the same provider. Extending this study to other providers and open-source models remains future work.

**Teacher model scope.** We use GPT-4o as the sole teacher. Section B.3 shows that the teacher's response style shapes the training data independently of task coverage. An open direction is self-distillation: using the student model itself as the teacher, recovering its failures via per-scenario optimization, and fine-tuning on its own corrected trajectories. Whether this preserves native response behavior while expanding task coverage, compared to cross-model distillation, is an open research question.

**Instruction-following assumption.** The load-bearing assumption of the method is that the executor can follow long, structured cheatsheets: the cheatsheets that drive recovery reach 7–9K characters with explicit step numbering, tool-call patterns, and conditional branches (Appendix D). A capable frontier executor (GPT-4o) satisfies this, but the assumption becomes binding for the self-distillation extension above, where a 4B student must execute the same frontier-written cheatsheets; its in-context instruction-following capacity sets an upper bound on how many failures self-distillation can recover. We have not measured this fraction directly; running frontier-written cheatsheets through the student executor would quantify the assumption and is a prerequisite for that extension.

**Benchmark constraints.** Both $\tau$2-bench and BFCL are designed as evaluation benchmarks; we repurpose their executors and verifiers into a data generation pipeline with separate train/eval splits. This limits data scale: only $\tau$2-bench telecom has sufficient task volume (2,285 tasks) for our experiments, while airline and retail are too small for end-to-end validation. On BFCL, tasks span 8 API domains and 4 evaluation categories, but each category contains only 200 tasks (800 total): the model must acquire multi-domain, multi-category tool-calling capabilities from very few examples per category. Per-category results at $n$=60 (test split, 70/30) reflect this: `miss_func` (which tests a capability the baseline largely lacks) shows consistent gains for both models, while other categories show mixed effects that are directional rather than definitive.

## Ethics Statement

To the best of our knowledge, all results reported in this paper are accurate. The two public benchmarks we use, $\tau$2-bench and BFCL, are openly available and cited accordingly, as are the teacher, reflector, and student models.

The production study in Section 4 is evaluated on an internal held-out set of synthetic scenarios that simulate production use cases. These scenarios are authored and human-reviewed for evaluation purposes; they are not sampled from real user traffic and contain no personally identifiable information. No human subjects were involved, and no private user data was used at any stage of data generation, training, or evaluation. Absolute performance values for the production system are withheld per deployment policy, so we report percentage-point gains over the respective baselines instead.

## Acknowledgments

We thank Frankie Liu for contributions to the exploratory experiments in this study and for extensive work on the internal product. We are grateful to Sylvia Xu for detailed feedback on multiple drafts of this paper. We also thank Anatoly Adamov, Alex Braunstein, and Amar Subramanya for their continued support of both the research and its path into production.

## References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. 2024. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pages 21246–21263.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Mohammad Hossein Amani, Aryo Lotfi, Nicolas Mario Baldwin, Samy Bengio, Mehrdad Farajtabar, Emmanuel Abbe, and Robert West. 2026. RL for reasoning by adaptively revealing rationales. *Preprint*, arXiv:2506.18110.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. $\tau^2$-Bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. FireAct: Toward language agent fine-tuning. *Preprint*, arXiv:2310.05915.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 10818–10838. PMLR.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: efficient finetuning of quantized LLMs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Google DeepMind. 2026. google/gemma-4-E4B-it. Hugging Face model card. Accessed: 2026-04-28.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. 2023. Reinforced self-training (ReST) for language modeling. *Preprint*, arXiv:2308.08998.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-STaR: Training verifiers for self-taught reasoners. In *Proceedings of the 2024 Conference on Language Modeling*.

Yuxuan Jiang, Dawei Li, and Francis Ferraro. 2026. DRP: Distilled reasoning pruning with skill-aware step decomposition for efficient large reasoning models. *Preprint*, arXiv:2505.13975.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, volume 2024, pages 54928–54958.

Yang Li, Youssef Emad, Karthik Padthe, Jack Lanchantin, Weizhe Yuan, Thao Nguyen, Jason Weston, Shang-Wen Li, Dong Wang, Ilia Kulikov, and Xian Li. 2025. NaturalThoughts: Selecting and distilling reasoning traces for general reasoning tasks. *Preprint*, arXiv:2507.01921.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems*, volume 37, pages 54463–54482. Curran Associates, Inc.

Yinyi Luo, Yiqiao Jin, Weichen Yu, Mengqi Zhang, Srijan Kumar, Xiaoxiao Li, Weijie Xu, Xin Chen, and Jindong Wang. 2026. AgentArk: Distilling multi-agent intelligence into a single LLM agent. *Preprint*, arXiv:2602.03955.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

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. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin

Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, and 21 others. 2024. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*. Expert Certification.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, Bangkok, Thailand. Association for Computational Linguistics.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, volume 2024, pages 30745–30766.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pages 12028–12068.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. 2025. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32600–32616, Vienna, Austria. Association for Computational Linguistics.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. Agent-R: Training language model agents to reflect via iterative self-training. *Preprint*, arXiv:2501.11425.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *Preprint*, arXiv:2308.01825.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: self-taught reasoner bootstrapping reasoning with reasoning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Zhiyuan Zhai, Wenjing Yan, Xiaodan Shao, and Xin Wang. 2026. Does RL expand the capability boundary of LLM agents? a PASS@(k,T) analysis. *Preprint*, arXiv:2604.14877.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. xLAM: A family of large action models to empower AI agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11583–11597, Albuquerque, New Mexico. Association for Computational Linguistics.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

# A Detailed Benchmark Results

This appendix expands the per-question summary in Section 4 with the full breakdowns it refers to: the failure analysis the method targets (A.1), recovery by reflector and convergence (A.2), the frontier-model comparison (A.3), and the post-training results (A.4).

## A.1 Failure Analysis

We characterize the failures that per-scenario optimization targets. On the $\tau$2-bench base split (50 airline, 114 retail, 114 telecom; 3 trials each, 834 total), 473 trials fail. Of these, 67% execute over half of the required tool calls correctly, yet only 7% reach the correct environment state and 6% produce the correct database state (Figure 2). Tool call accuracy ranges from 51% in airline to 71% in telecom.

We confirmed this at larger scale: on 2,285 telecom tasks (1 trial each), 2,100 fail. Of these, 71% get over half of tool calls right, while environment and database correctness drop to 3% and 0.1%.

These failures share a common structure: the teacher completes routine operations correctly, then misses a single action at a decision point that depends on state accumulated earlier in the trajectory. Because $\tau$2-bench evaluates end-to-end correctness, one missed tool call cascades into incorrect environment and database state, and the entire trajectory is discarded. Figure 5 shows three

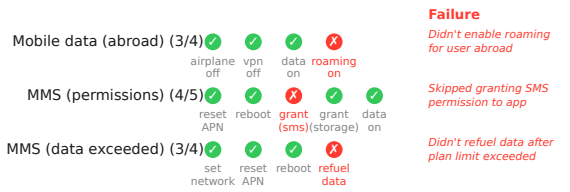


Figure 5: Three failed GPT-4o trajectories from $\tau$2-bench telecom. Green circles indicate tool calls that matched the ground truth; red circles indicate missed calls. Each trajectory completes most required actions but fails at one decision point, causing the entire trajectory to be discarded by the filter.

concrete examples: a skipped roaming toggle for a customer abroad, a missing SMS permission grant for an MMS issue, and an overlooked data refuel after the plan limit was exceeded. In each case, 3 or 4 out of 4–5 required actions were correct.

## A.2 Data Generation

**Reflection model capability drives recovery rate.** Recovery effectiveness depends on the reflection model that analyzes failed trajectories and generates cheatsheets. We ablated four reflection models on 82 scenarios that failed all 3 trials under GPT-4o across three $\tau$2-bench domains (Figure 3). All other variables are held constant: the agent model (GPT-4o, temp=0), user simulator (GPT-4.1, temp=0), and optimizer configuration (max 10 iterations, stop at reward=1.0).

Reflection model capability is the dominant factor. GPT-5.1 recovers 97.6% of failed scenarios (80/82) versus 24.4% for GPT-4o (20/82, self-reflection). The gap is largest on telecom, the most complex domain (4/55 vs. 54/55). Recovery counts include both cheatsheet-driven improvements and cases where the teacher succeeded on stochastic replay without a cheatsheet; Appendix F reports the breakdown.

**Convergence behavior.** On the 300-task telecom production run with GPT-5.1 as reflector, convergence is bimodal (Figure 6). Of the 300 attempted failures, 279 are recovered (264 driven by a cheatsheet, 15 by stochastic replay without one) and 21 are never recovered. Of the cheatsheet-driven recoveries, 58.7% resolve on the first iteration, 76.5% by the second, and 84.1% by the third. Scenarios that remain unsolved after 3 iterations rarely recover in later iterations. This suggests that the mechanism is cheatsheet *quality* (whether the reflection correctly diagnoses the root cause) rather than brute-force search over prompt variations.

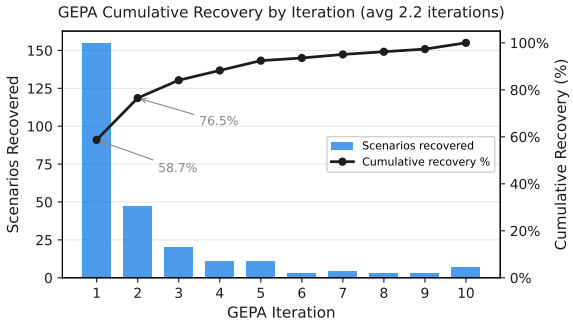


Figure 6: Cumulative recovery by optimizer iteration on 300 failed telecom tasks (GPT-4o agent, GPT-5.1 reflector). 264 of 279 total recoveries are cheatsheet-driven; the remaining 15 are stochastic replays. Most solvable scenarios resolve within 2–3 iterations.

| Agent | Reflection | Recovered | Rate |
|---|---|---|---|
| GPT-5.1 (no reasoning) | none | 44/300 | 14.7% |
| GPT-5.1 (high) | none | 250/300 | 83.3% |
| GPT-4o | GPT-5.1 (high), 10 iter | 279/300 | 93.0% |
| GPT-5.1 (high) | GPT-5.1 (high), 1 iter | 295/300 | 98.3% |

Table 3: Recovery rate on 300 failed telecom tasks (selected from GPT-4o failures by design). GPT-5.1 defaults to reasoning_effort=none (no reasoning); "high" = reasoning_effort=high. Per-scenario optimization with a weaker agent (GPT-4o) and capable reflector recovers 93.0% using the expensive model only for reflection (avg 2.2 calls per scenario vs. ∼11.5 execution turns per task for direct generation).

**Data composition.** The composition of the combined training data differs sharply between the two benchmarks. On $\tau$2-bench, recovered trajectories constitute 64% of the combined set, so training signal comes predominantly from tasks the teacher originally failed. On BFCL, recovered trajectories constitute only 23%; the dataset remains composed primarily of tasks the teacher already solves. Appendix J reports the full trajectory counts and example volumes per benchmark and category.

## A.3 Can Frontier Models Replace Per-Scenario Optimization?

The capable model used as the reflector (GPT-5.1) could also generate trajectories directly, without the per-scenario optimization loop. We compare four generation strategies on 300 telecom tasks that GPT-4o failed in the initial generation pass (Table 3).

**GPT-5.1 without reasoning: 14.7% recovery.** GPT-5.1 without reasoning solves 44 of 300 tasks (14.7%). Upgrading the model alone (without enabling reasoning) helps over GPT-4o, but leaves 85% of failures unresolved.

**GPT-5.1 at high reasoning: 83.3% recovery.** GPT-5.1 at high reasoning effort solves 250 (83.3%). This approaches the recovery rate of per-scenario optimization with a weaker agent (GPT-4o agent + GPT-5.1 reflector: 279/300, 93.0%), but requires the most expensive inference setting on every turn of every task.

**Adding one reflection step: 98.3% recovery.** Adding a single reflection step to GPT-5.1 high pushes recovery to 295/300 (98.3%). The strongest model we tested still benefits from per-scenario cheatsheet guidance, producing 45 additional recoveries beyond what high-reasoning direct generation achieves alone.

**Model usage pattern.** In the GPT-4o executor configuration (row 3), the capable model is used only for reflection (2.2 calls per scenario on average; 58.7% resolve on the first cheatsheet), while the weaker model handles multi-turn execution. Frontier direct execution at full reasoning budget (row 2) requires the capable model on every turn of the task. The two strategies also produce stylistically different trajectories; Section B.3 examines how the executing model's behavior shapes training data.

### A.4 Post-Training Results

Figure 4 summarizes the main results across both models and benchmarks. Each model is evaluated under three conditions: no fine-tuning (baseline), QLoRA on filtered-only data, and QLoRA on the combined dataset (filtered plus recovered trajectories).

**$\tau$2-bench.** On $\tau$2-bench telecom (Figure 4, left), Qwen3-4B-Instruct-2507 improves from a baseline of 0.132 to Pass^1 = 0.529 with combined training.[3] For context, the GPT-4o teacher that generated the training data scores 0.205 on the same 114-task evaluation set under the same protocol (3 trials, temp=0). Gemma 4 E4B-it presents a harder test: its baseline (0.231) is already on par with the GPT-4o teacher. The method still improves it to 0.333: although GPT-4o executes the trajectories, the recovery process is guided by GPT-5.1 reflection, so the training signal encodes capability beyond what the executing teacher produces alone. Action accuracy follows the same pattern: the combined model reaches 87.3% (Qwen) and 83.3% (Gemma), up from 60.9% and 41.4% respectively. Section A.5 tests whether these improvements hold under stochastic sampling at temperature 1.

**BFCL v4 multi-turn.** On BFCL v4 multi-turn (Patil et al., 2025), Gemma 4 E4B-it gains +7.2 percentage points overall, while Qwen3-4B-Instruct-2507 gains +1.9pp. Filtered-only training improves partial rewards but does not improve end-to-end task completion over the baseline in any of the four model–benchmark pairs. Turn-level partial reward confirms the pattern: Gemma improves from 0.181 to 0.286 (+58%); Qwen's partial reward is flat (0.316 to 0.301).[4] Section B.2 examines why. The gain concentrates in the `miss_func` category (recognizing unavailable functions), the only category where both models improve consistently; per-category results are in Appendix L.

### A.5 Sampling Robustness

The improvements reported above use greedy decoding (temp=0). To test whether training gains reflect genuine capability expansion rather than improved greedy-path reliability, we follow the framework of Zhai et al. (2026). We identify "flip tasks" (tasks that changed from 0/3 passes at baseline to $\geq$2/3 after fine-tuning) and re-evaluate both models on these tasks at temperature 1.0 with 10 trials per task. Flip tasks are post-hoc selected on temp=0 gains, so this analysis measures robustness on the subset where fine-tuning helped, not overall capability; the subsets are small (Qwen $n$=46, Gemma $n$=21).

Figure 7 shows per-task pass rates on the flip tasks. On these tasks the combined model sustains substantial per-trial success under stochastic sampling: Qwen averages 54.1% (46 tasks) and

[3] We verified training stability by retraining from scratch and re-evaluating multiple times. Across independent training runs and evaluations, Qwen Pass^1 ranged from 0.518 to 0.538, confirming that the reported gains are not artifacts of a favorable random seed. See Appendix N for details.

[4] We ran three independent evaluations per condition to assess serving non-determinism. Overall accuracy is stable across runs (spread $\leq$1.6pp); per-category results (60 tasks each) vary by up to 3.3pp from this source alone, which should be considered when interpreting category-level comparisons.

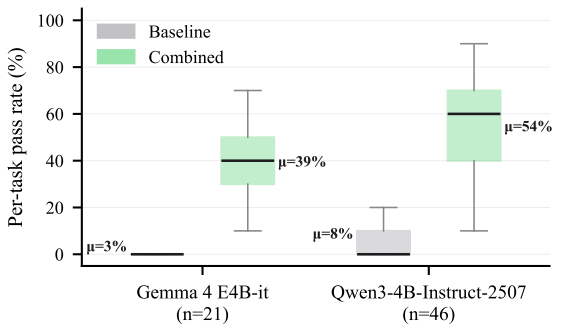


Figure 7: Distribution of per-task pass rates at temperature 1 (10 trials) on $\tau$2-bench flip tasks. Boxes show quartiles; black lines mark means. Baseline rates cluster near zero; combined rates are broadly distributed, confirming that training gains persist under stochastic sampling.

Gemma 39.0% (21 tasks), versus near-zero baselines (8.3% and 2.9% respectively). The pattern is consistent across individual tasks: the combined model produces broadly distributed success rates while the baseline is near zero almost everywhere.

## B Additional Benchmark Analyses

### B.1 Data Quality

**Cheatsheet integrity: the scaffold does not create shortcuts.** A natural concern is whether cheatsheets introduce unrealistic shortcuts into the generated trajectories. Because the cheatsheet is stripped before SFT, the student model never sees it. However, if the cheatsheet causes the teacher to bypass discovery steps (e.g., using an entity ID without first looking it up), the resulting trajectory would teach the student to hallucinate values rather than discover them through tool calls.

We audited 90 cheatsheet-guided trajectories: 60 from $\tau$2-bench (15 per reflection model across all four reflectors) and 30 from BFCL (using the production reflector, GPT-5.4), using two independent LLM judges (Claude Opus 4.6 and Claude Sonnet 4.6). Each judge evaluated the full, untruncated trajectory against three checks: (1) no hardcoded values (all tool call arguments sourced from the user's request, prior tool returns, or schema defaults); (2) no magic numbers (computed values derived from prior lookups, not hardcoded); and (3) no future information (each tool call uses only information available at the time it was called). Table 4 reports the results.

Both judges agree that all 90 trajectories preserve value discovery through tool calls (97.8% inter-judge agreement). Two edge cases received split verdicts; manual inspection confirmed no actual leakage (details in Appendix C).

| Reflection Model | Opus 4.6 | Sonnet 4.6 |
|---|---|---|
| *$\tau$2-bench (60 samples, 15 per teacher)* | | |
| GPT-4o | 15/15 | 14/15 |
| GPT-5-mini | 15/15 | 15/15 |
| GPT-4.1 | 15/15 | 15/15 |
| GPT-5.1 | 15/15 | 15/15 |
| *BFCL v4 (30 samples, GPT-5.4 teacher)* | | |
| All categories | 30/30 | 29/30 |
| **Total** | **90/90** | **88/90** |

Table 4: Cheatsheet integrity audit. Each cell shows the number of trajectories rated TRANSFERABLE (all discovery steps preserved). Sonnet's two flags are false positives on manual inspection (Appendix C).

Qualitatively, cheatsheets teach *process* (troubleshooting methodology, tool ordering, and error recovery patterns) rather than *answers*. Stronger reflection models produce longer, more structured cheatsheets that describe procedures without referencing scenario-specific values, while GPT-4o's shorter cheatsheets occasionally include task-specific hints alongside discovery instructions. While this style difference does not compromise trajectory integrity (discovery instructions are present across all reflection models), it may contribute to the recovery rate gap observed in Section A.2: more structured cheatsheets appear to guide the teacher more reliably past failure points.

### B.2 Fine-Tuning Analysis

**Reward breakdown.** $\tau$2-bench evaluates task success using a configurable set of criteria: database checks, environment/status assertions, action matching, communication-information checks, and natural-language assertions. In telecom, task success is determined primarily by programmatic environment assertions, with action matching included for transfer-style tasks; communication checks and NL assertions are not used. This makes telecom evaluation largely programmatic and avoids variance from LLM-judged criteria. Table 5 reports action, environment, and database metrics as diagnostic breakdowns for the telecom domain.

Filtered-only training improves action accuracy by +25pp but leaves environment assertions and database accuracy flat. Combined training closes this gap: environment assertions improve by

| Experiment | Action | Env | DB |
|---|---|---|---|
| Baseline (no SFT) | 41.4% | 23.5% | 9.3% |
| + QLoRA Filtered | 66.3% | 22.6% | 10.9% |
| + QLoRA Combined | **83.3%** | **54.5%** | **15.6%** |

Table 5: Reward breakdown on $\tau$2-bench telecom (base-114 held-out tasks, 3 trials, temp=0). Filtered-only training improves action accuracy but not environment or database accuracy. Combined training improves all three, with the largest gain in environment assertions.

| Metric | GPT-4o | GPT-5.1 (high) |
|---|---|---|
| Trajectories | 259 | 259 |
| Text responses | 3,180 | 2,863 |
| Avg. length (chars) | 308 | 987 |
| Median length (chars) | 253 | 965 |
| Tool calls / trajectory | 5.2 | 5.2 |

Table 6: Text response characteristics on 259 matched telecom tasks. GPT-4o uses cheatsheet guidance; GPT-5.1 uses direct high reasoning. Tool call counts are comparable; response length differs by 3.8×.

+31pp and database accuracy by +6pp. The recovered trajectories expand task coverage to scenarios the teacher originally failed, adding demonstrations of tasks that require multi-step state management beyond what the teacher handles on the first attempt. Appendix L reports a parallel category-level breakdown for BFCL.

### B.3 Teacher Response Complexity

GPT-5.1 at high reasoning matches or exceeds GPT-4o's recovery rate (Table 3), but the resulting trajectories differ in structure. A natural question is why not use GPT-5.1 directly as the teacher, bypassing the recovery loop entirely. We investigate by comparing GPT-4o and GPT-5.1 trajectories on matched tasks.

**Controlled comparison on matched tasks.** From the 300-task recovery set, 15 tasks pass on GPT-4o replay alone (no cheatsheet; stochastic user variation), 264 require cheatsheet guidance, and 21 are never recovered. GPT-5.1 (high reasoning, no scaffold) passes 295 of these tasks directly. We compare text responses on the 259 tasks both models solve; because task identity is held constant, differences in response characteristics reflect model behavior rather than task difficulty. Figure 8 and Table 6 report text response statistics on this matched set, with the 15 replay-pass tasks as a separate condition in the figure.

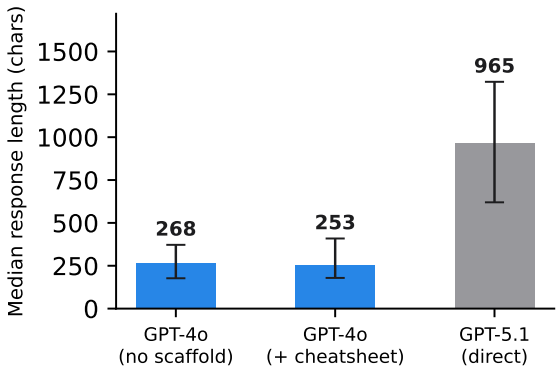


Figure 8: Median text response length on matched tasks. GPT-4o without scaffold (n=15 tasks) and GPT-4o with cheatsheet (n=259) produce comparable response lengths, while GPT-5.1 responses are 3.8× longer. Whiskers show IQR.

GPT-5.1 text responses are 3.8× longer at the median (965 vs. 253 characters), despite comparable tool call counts (5.2 calls per trajectory). GPT-4o produces more text turns per trajectory (12.3 vs. 11.1), each substantially shorter: it confirms the completed action in one sentence, suggests one next step, and waits for the user to report back. GPT-5.1 front-loads a numbered multi-step protocol into a single turn, with specific device instructions and a conditional branching plan. Both approaches resolve the task; the verifier checks final state and required actions, not response phrasing. Appendix H shows both models on the same scenario at a representative 3.7× ratio.

**Scaffold effect on response style.** Because per-scenario optimization uses GPT-4o as the executing agent, its trajectories preserve GPT-4o's incremental communication pattern even on tasks that required GPT-5.1's reflection to recover. Figure 8 and the per-tool breakdown in Appendix H show that the cheatsheet changes what tools GPT-4o calls but does not alter how it communicates results: GPT-4o replay passes (no cheatsheet, n=15) show a median of 268 characters, comparable to GPT-4o with cheatsheet at 253 (0.94×). Neither condition approaches GPT-5.1's 965-character median.

This separates two concerns in data generation: task coverage (determined by the reflector model) and response style (determined by the executing agent), which can in principle be varied independently.

## C Cheatsheet Integrity Audit

### C.1 Judge Rubric

Each of the 90 trajectories was evaluated independently by Claude Opus 4.6 (`claude-opus-4-6`) and Claude Sonnet 4.6 (`claude-sonnet-4-6`) using the prompt in Figure 9. The judge receives the full, untruncated task definition, cheatsheet, and trajectory as input. The three checks are summarized below; the exact prompt wording is reproduced for full reproducibility.

**Check 1: No hardcoded values.** For every value used in a tool call argument (entity IDs, file paths, state values, amounts), was it either (a) provided in the user's request, (b) returned by a prior tool call, or (c) a reasonable default from the tool schema? Answer "no" only if a value could not have been obtained without reading the cheatsheet.

**Check 2: No magic numbers.** If the trajectory uses a computed value, does it include the steps to compute or look up that value? Answer "no" if the trajectory uses a hardcoded number that only makes sense given the cheatsheet.

**Check 3: No future information.** Does each tool call use information available at the time it was called? The question is whether information flows correctly, not whether the ordering is optimal or whether calls are sequential versus parallel.

**Verdict.** TRANSFERABLE or HAS_SHORTCUT. Parallel tool calls, action ordering preferences, missing user confirmations, and policy violations are explicitly excluded from consideration; these are execution style, not discovery bypass.

### C.2 Sonnet False Positive Analysis

Sonnet flagged 2 of 90 samples as HAS_SHORTCUT. Both are false positives on manual inspection.

**Sample 11 ($\tau$2-bench telecom, GPT-4o teacher).** After receiving line IDs [L1001, L1002, L1003] from the customer profile, the agent fetched only L1002 directly. Sonnet flagged this as skipping the phone number matching step. However, the customer's phone number (555-123-2002) shares a suffix pattern with L1002, making this a plausible heuristic. The tool return confirms the match (`phone_number: 555-123-2002`). The agent does not proceed without verification; it receives and acts on the confirmed match.

**Sample 77 (BFCL miss_func, GPT-5.4 teacher).** Sonnet flagged the use of Twitter credentials (`tech_guru`, `securePass123`) as cheatsheet-sourced. The user's message appears truncated in the trajectory representation at "My user name..." However, BFCL provides fixed, deterministic user messages, and the full message includes both username and password. The truncation is an artifact of the trajectory display format, not the actual conversation the agent processed.

### C.3 Cheatsheet Examples by Reflection Model

Representative cheatsheets from four reflection models on $\tau$2-bench telecom scenarios illustrate the relationship between teacher capability and cheatsheet style.

**GPT-4o (1.8K chars).** Short, mixes task-specific hints with discovery instructions. Example: "Identify relevant line: Use the correct line ID associated with the phone number, in this case, L1002." The discovery instruction is present but the specific value is named.

**GPT-5-mini (9.8K chars).** Verbose, focuses on behavioral patterns. Identifies wrong *behaviors* (e.g., "the assistant did not correctly refuel data") and prescribes corrective actions.

**GPT-4.1 (3.7K chars).** Mid-length, structured. Separates failure diagnosis from step-by-step guidance. Uses parameterized instructions without naming specific values.

**GPT-5.1 (7.2K chars).** Identifies wrong *assumptions* rather than wrong behaviors. Names core patterns explicitly (e.g., "User eligible but no change/cancel requested, core of this task"). Fully parameterized with no scenario-specific values.

## D Cheatsheet Examples

Figures 10 and 11 show representative cheatsheets from the weakest and strongest reflection models on $\tau$2-bench scenarios. Figure 12 shows a BFCL example. GPT-4o produces short, general guidance (1.8K chars); GPT-5.1 produces detailed, parameterized procedures (7–9K chars) with explicit step numbering, tool call patterns, and state tracking rules.

```
You are an expert reviewer analyzing whether a tool-calling trajectory teaches
transferable behavior, or whether it contains shortcuts that would break on a new task.

Context: A teacher model (GPT-4o) generated this trajectory with help from a
"cheatsheet" -- a natural-language hint appended to the system prompt. The cheatsheet
is stripped before SFT. The student never sees the cheatsheet.

The cheatsheet being prescriptive is fine and expected. The only concern is: does the
resulting trajectory teach a tool-calling pattern that would work on a new, similar task?

What is NOT a shortcut (do NOT flag these):
- Parallel/batched tool calls: execution style, not discovery bypass.
- Action ordering preferences: diagnostic choice, not a shortcut.
- Missing user confirmation: politeness issue, not discovery bypass.
- Policy violations: format issue, not transferability issue.

Only flag as HAS_SHORTCUT if a value used in a tool call argument was NOT discoverable
from the user's request or prior tool returns.

Inputs: [task_definition] [cheatsheet] [trajectory]

Checks (yes/no + rationale):
  1. No hardcoded values: all argument values sourced from user request, prior tool
     returns, or schema defaults? "no" ONLY if a value could not have been obtained
     without reading the cheatsheet.
  2. No magic numbers: computed values justified by prior steps? "no" if the trajectory
     uses a number that only makes sense if you read the cheatsheet.
  3. No future information: each tool call uses information available at call time?
     "no" ONLY if a tool call uses a value not yet returned by any prior call.

Verdict: TRANSFERABLE or HAS_SHORTCUT.
```

Figure 9: Exact prompt template given to both LLM judges. Placeholders `[task_definition]`, `[cheatsheet]`, and `[trajectory]` are filled with the full, untruncated text for each sample.

## E Hyperparameters

Both models use the same QLoRA configuration.

| Parameter | Value |
|---|---|
| LoRA rank | 32 |
| LoRA alpha | 32 |
| Quantization | 4-bit (QLoRA) |
| Learning rate | 2e-4 |
| LR scheduler | cosine |
| Warmup steps | 20 |
| Epochs | 3 |
| Batch size | 4 |
| Max sequence length | 16,384 |
| Weight decay | 0.1 |

Table 7: QLoRA training hyperparameters (shared across both models).

## F Recovery Breakdown

Per-scenario optimization recovery includes two sources: *improved* scenarios where the cheatsheet enabled the teacher to pass (initial_score $< 1$, best_score $\geq 1$), and *replay* scenarios where the teacher passed on a stochastic re-execution without a cheatsheet (initial_score $\geq 1$). Table 8 reports the breakdown for the 82-task cross-domain reflector study.

## G Missing Functions (`miss_func`)

BFCL v4 multi-turn includes a `miss_func` category that tests two capabilities (Patil et al., 2025): (1) the model must identify that no available function can fulfill a user request, and (2) once the missing functions are provided in the next turn, the model must use them. Functions are held out from the tool list at the start and silently added back at a specific "holdout turn," where the user says "I have updated some more functions you can choose from. What about now?"

This is the hardest category for all models tested. The pass rates here are measured over the full 200-task `miss_func` category (this deep-dive uses the full category for statistical power; the headline per-category results in Table 12 use the 60-task test split): GPT-4o passes 43%, Qwen3-4B-Instruct-2507 passes 10.5% (21 of 200), and Gemma 4 E4B-it passes 0.5% (1 of 200). On the 60-task test split, the corresponding baselines are 11.7% (Qwen) and 0.0% (Gemma; Table 12).

```
## TASK CHEATSHEET
## Handling Exchanges and Cancellations

<failures>
- Attempting to exchange items not marked as delivered.
- Multiple tool calls executed simultaneously.
- Failure to confirm refund and payment options with user before proceeding.
</failures>

<guidance>
   <steps>
   - Authenticate the user first using their email, or name and zip code.
   - Verify the status of each order pertaining to the request.
   - Ensure the items for exchange are part of a delivered order.
   - Collect all necessary details including user confirmation for each task.
   - For exchanges, confirm items to be exchanged and check availability.
   - Ensure single tool call execution at a time.
   - Confirm with the user their choice of payment method before processing.
   - List all intended actions and gain user confirmation before executing.
   </steps>
</guidance>
```

Figure 10: Full cheatsheet generated by GPT-4o (self-reflection) for a $\tau$2-bench retail exchange scenario. The guidance is general and mixes procedural advice with policy reminders. Discovery instructions are present ("verify the status," "check availability") but brief.

| Reflector | Domain | Improved | Replay | Total | Rate |
|---|---|---|---|---|---|
| GPT-4o | airline | 4 | 1 | 5/13 | 38.5% |
| | retail | 9 | 2 | 11/14 | 78.6% |
| | telecom | 3 | 1 | 4/55 | 7.3% |
| | overall | 16 | 4 | 20/82 | 24.4% |
| GPT-5-mini | airline | 9 | 1 | 10/13 | 76.9% |
| | retail | 9 | 3 | 12/14 | 85.7% |
| | telecom | 27 | 0 | 27/55 | 49.1% |
| | overall | 45 | 4 | 49/82 | 59.8% |
| GPT-4.1 | airline | 8 | 2 | 10/13 | 76.9% |
| | retail | 9 | 5 | 14/14 | 100% |
| | telecom | 44 | 0 | 44/55 | 80.0% |
| | overall | 61 | 7 | 68/82 | 82.9% |
| GPT-5.1 | airline | 12 | 0 | 12/13 | 92.3% |
| | retail | 13 | 1 | 14/14 | 100% |
| | telecom | 52 | 2 | 54/55 | 98.2% |
| | overall | 77 | 3 | 80/82 | 97.6% |

Table 8: Recovery breakdown for the 82-task reflector ablation. *Improved*: cheatsheet enabled the teacher to pass. *Replay*: teacher passed on stochastic re-execution without a cheatsheet. Replay accounts for 4–7 of the total recoveries per reflector; the majority of recoveries are cheatsheet-driven.

**Where does Gemma fail?** We analyzed 50 tasks from the full category to determine which step causes failure. On the turn *before* the holdout (when the requested function is missing), Gemma uses a different tool instead of indicating inability in 72% of cases (36/50). On the holdout turn itself (when the missing function is added back), Gemma fails to call the newly available function in 96% of cases (48/50), producing a text response instead of a tool call.

By contrast, GPT-4o also uses a different tool on the pre-holdout turn (96%, 47/49), but succeeds on the holdout turn 76% of the time (37/49). Both models rarely indicate inability on the pre-holdout turn; the primary differentiator is whether the model calls the newly added function when prompted.

**Offline replay.** We replayed the holdout turn for `miss_func_1` using the `google/gemma-4-E4B-it` checkpoint locally with the same chat template used in our vLLM evaluation pipeline. We verified from the BFCL inference logs that the tool list grows from 17 to 18 functions on the holdout turn, with `mv` appended. The model's system prompt includes all 18 tools in `<|tool>` declarations. The user message on the holdout turn is: "I have updated some more functions you can choose from. What about now?" It does not name which function was added. The model produces:

> "I'm ready for the new functions! Please provide them, and I'll be happy to see what I can do with them."

The model generates text instead of a tool call.

**Ablation: explicit vs. vague prompt.** To test whether the failure is due to the model's inability to use newly added tools or the vagueness of the holdout prompt, we replaced the BFCL default message ("I have updated some more functions you can choose from. What about now?") with an explicit instruction: "I have added a sort

```
## TASK CHEATSHEET
## MMS Issues When User Is Abroad, With Possible Data Limit,
## Roaming, and Device Misconfiguration

<failures>
- Focusing only on device-side MMS troubleshooting and ignoring
  account-level causes (data limit reached, roaming disabled).
- Selecting the wrong line in a multi-line account.
- Transferring to human agent before exhausting all steps.
</failures>

<guidance>
   <steps>
   - 1. Identify the customer and the correct line
     - Call get_customer_by_phone(phone_number=<user_phone>).
     - For each line_id, call get_details_by_id(id=<line_id>).
     - Select the line where phone_number matches the user's number.
   - 2. Gather plan and usage information
     - Call get_details_by_id(id=<plan_id>) for data_limit_gb.
     - Call get_data_usage(customer_id=..., line_id=...).
   - 3. Run basic MMS/device connectivity checks
     - If status bar shows 2G, set_network_mode_preference("4g_5g").
   - 4. Check Wi-Fi Calling (toggle off if ON)
   - 5. Verify messaging app permissions (grant sms, storage)
   - 6. Handle data limit (refuel_data up to 2GB with cost confirm)
   - 7. Handle roaming (enable_roaming + device toggle_roaming)
   - 8. Test can_send_mms() after each major change
   - 9. Tool-call discipline: one call per turn, no mixing
   - 10. Transfer to human only after exhausting all steps
   </steps>
</guidance>
```

Figure 11: Abbreviated cheatsheet generated by GPT-5.1 for a $\tau$2-bench telecom MMS scenario (full version: 8.8K chars). The guidance is parameterized ("<user_phone>", "<line_id>") with no hardcoded IDs. Each step specifies which tool to call and what to do with its return value.

function to your available tools. Please sort the file now." With this change, the model correctly produces a tool call to the newly added function. This confirms that Gemma can use tools added mid-conversation when the instruction names the function and the action; the failure on `miss_func` is specific to the vague holdout prompt, which does not indicate which function was added or what action to take.

**Effect of fine-tuning.** After fine-tuning with combined data, Gemma reaches 21.5% on `miss_func` (43 of 200). The fine-tuned model learned to respond with tool calls on holdout turns rather than text, reducing the text-only response rate from 85.5% to 22.0% of turns on this category.

## H Trajectory Communication Patterns

We present text responses from GPT-4o and GPT-5.1 (high reasoning) on the same task, both without any cheatsheet scaffold (replay passes). Both models have just called `get_details_by_id`, received structured account information confirming the line is active and the issue is phone-side, and now communicate next steps to the user. The task is MMS troubleshooting where the user's mobile data is off, Wi-Fi calling interferes, and the SIM is unseated.

### GPT-4o, no cheatsheet (305 characters).

> I've found your account and confirmed that your line is active. Let's proceed with troubleshooting the MMS issue. Could you please check if your phone has cellular service and mobile data enabled? You can do this by checking the status bar on your phone for signal strength and data connection indicators.

### GPT-5.1 high reasoning, no cheatsheet (1,132 characters).

> I've pulled up your line ending in 2002. Your line is active, roaming is enabled, and you still have data left in your plan, so nothing on the account/plan side is blocking MMS. Let's check the phone side next.
>
> First I want to confirm that **mobile data itself is working**, since MMS depends on that.
>
> Please do this and tell me the results:
>
> **1. Turn off Wi-Fi completely**
> Swipe down from the top and tap the Wi-Fi icon so it's OFF.

```
## TASK CHEATSHEET

<failures>
- The model tried to start the engine immediately after converting liters
  to gallons, but VehicleControlAPI required prerequisite state changes.
- The model missed required steps: locking all doors, pressing the brake
  pedal, then starting the engine.
- The model used the healthy_tire_pressure field from check_tire_pressure,
  but the task-defined rule must override: healthy means all four tire
  pressures are between 32 and 35 inclusive.
</failures>

<guidance>
<steps>
1. If the user does not provide the liter amount, ask for it first.
2. Once provided, call liter_to_gallon({"liter": <number>}).
3. For VehicleControlAPI, perform the required setup sequence:
   - lockDoors({"unlock": false,
       "door": ["driver","passenger","rear_left","rear_right"]})
   - pressBrakePedal({"pedalPosition": 1})
   - startEngine({"ignitionMode": "START"})
4. For tire pressure: call check_tire_pressure({}).
   - Healthy only if every tire is between 32 and 35 inclusive.
5. For tweet: post_tweet({"content":"healthy"|"not healthy",
     "tags":["#CarMaintenance"],"mentions":["@VehicleGuru"]})
6. For retweet: use the tweet id returned by post_tweet.
</steps>
</guidance>
```

Figure 12: Abbreviated BFCL cheatsheet (GPT-5.4 teacher, `miss_param` category). The cheatsheet teaches API prerequisite sequencing (lock doors before engine start) and evaluation-specific rules (tire health threshold). All argument values are schema constants or user-provided; no task-specific IDs are hardcoded.

> **2. Make sure Mobile Data is ON**
> In the same quick settings area, make sure "Mobile data" is turned ON.
>
> **3. Test mobile data**
> With Wi-Fi still OFF, open your browser and try to load any website.
>
> **4. Tell me:**
> Does the website load or not? When you try to send a picture message right now (with Wi-Fi still OFF and mobile data ON), does it stay stuck on "sending," show a specific error message, or fail immediately?
>
> Once you tell me what happens in steps 3 and 4, I'll guide you through the next specific checks (network type/APN/Wi-Fi Calling/app permissions) based on your result.

**Observations.** Both models have received the same account information from `get_details_by_id` and both intend the same next action: verify that the user's mobile data connection works. GPT-4o makes a single request and waits (305 characters). GPT-5.1 first summarizes what it learned from the tool return, then provides a numbered protocol with specific UI instructions and a conditional branching plan for what comes next (1,132 characters; 3.7× longer).

This illustrates the interaction style difference reported in Section B.3: GPT-4o sequences diagnostics one at a time across turns (incremental), while GPT-5.1 front-loads a multi-step plan into a single turn (batch). Both strategies resolve the task successfully (reward = 1.0). The difference is not in the number of tool calls (comparable on average across the matched set) but in how much context each model provides per text turn.

**Task and provenance.** Task ID: `[mms_issue]bad_wifi_calling|data_mode_off|unseat_sim_card[PERSONA:Easy]`. Both trajectories are from independent generation runs on the same task definition without cheatsheet guidance (improvement_source = replay_only). This example is near the median of the length ratio distribution (3.7× vs. 3.8× median on 259 matched tasks).

**GPT-4o with cheatsheet vs. GPT-5.1 direct.** The example above uses replay-only passes (no cheatsheet) to isolate intrinsic model style. We now show that GPT-4o with cheatsheet guidance exhibits the same incremental pattern. The following responses are from a cheatsheet-recovered task in the 259-task matched set, both after calling `get_customer_by_phone` and identifying the user's line.

**GPT-4o + cheatsheet (299 characters).**

> Thank you for the information. I've identified your line associated with the phone number 555-123-2002. Let's proceed with troubleshooting the MMS issue.
>
> Could you please check if your phone can send MMS messages by running the `can_send_mms()` diagnostic on your device and let me know the result?

**GPT-5.1 direct, no cheatsheet (1,141 characters).**

> Please try them now, one by one. That will give us clear answers:
>
> **1. Turn off Wi-Fi first** so we're only testing mobile data.
> **2. Then, in this order, tell me what happens for each:**
>
> - Regular call: Try calling any number. Does it connect and stay connected?
> - Regular SMS text (no picture): Send a plain text to someone. Does it send?
> - Internet over mobile data: With Wi-Fi still OFF, open a browser. Does it load normally, slowly, or not at all?
> - MMS (picture message): Try sending a photo. Does it stay on "sending," show "failed," or something else?
>
> Also, please look at the top of your screen and tell me what network type/icon you see (e.g., 4G, LTE, 5G, E, H, R) and whether there's any roaming icon.
>
> Once you report back with these results, I'll run checks on your line and walk you through the next steps.

**Observations.** GPT-4o with cheatsheet produces the same incremental pattern as GPT-4o without cheatsheet: identify the line, request one diagnostic, wait (299 characters). GPT-5.1 requests a full test battery covering calls, SMS, data, and MMS in a single turn (1,141 characters; 3.8× longer). The cheatsheet specifies diagnostic ordering (which tools to call and in what sequence), not response verbosity or framing; GPT-4o's communication style remains unchanged even on tasks it originally failed. Task ID: `[mms_issue] airplane_mode_on|...|data_usage_exceeded| user_abroad_roaming_enabled_off [PERSONA:None]`.

**Per-tool response length breakdown.** The qualitative examples above use replay-only and cheatsheet-guided trajectories to illustrate the style difference. Table 9 below reports aggregate statistics on the full 259-task matched set. The ratios are consistent across tool types and across both conditions, confirming that the cheatsheet does not alter GPT-4o's communication pattern.

| After tool | GPT-4o (+cheatsheet) | GPT-5.1 (direct) | Ratio |
|---|---|---|---|
| `get_details_by_id` | 469 | 997 | 2.1× |
| `enable_roaming` | 292 | 1,254 | 4.3× |
| `get_data_usage` | 476 | 1,450 | 3.0× |
| `refuel_data` | 206 | 835 | 4.1× |
| All text responses | 308 | 987 | 3.2× |

Table 9: Average text response length (characters) after each tool call type on 259 matched tasks. The ratio is consistent across tool types (2.1–4.3×), confirming the style difference is systematic rather than specific to particular actions.

| | |
|---|---|
| **Gemma 4 E4B-it** | |
| Parameters | 4.5B effective (8B total) |
| Architecture | Multimodal (text, image, audio) |
| Layers / Hidden | 42 / 2,560 |
| Attention | 8Q / 2KV, hybrid sliding+full |
| Context | 128K tokens |
| Tool format | Gemma special tokens |
| **Qwen3-4B-Instruct-2507** | |
| Parameters | 4.0B (3.6B non-emb.) |
| Architecture | Text-only causal LM |
| Layers / Hidden | 36 / 2,560 |
| Attention | 32Q / 8KV, full |
| Context | 262K tokens |
| Tool format | JSON in XML tags |

Table 10: Student model comparison. Both share a hidden dimension of 2,560 and operate at comparable effective scale but differ in architecture, modality, and tool-call format.

## I Student Model Comparison

Table 10 summarizes the two student models. Both operate at comparable effective scale (∼4B parameters) but differ in architecture, modality, and tool-call format. Training on both and observing consistent improvements across the two confirms that the method generalizes across model families.

## J Training Data Composition

Table 11 reports the training data composition for both benchmarks. Each multi-turn trajectory is unrolled into prompt/completion pairs (one per assistant turn).

## K BFCL Turn-Level Partial Reward

BFCL v4 multi-turn evaluates tasks as binary pass/fail (all turns must pass for the task to succeed). The benchmark does not provide a built-in partial reward metric. We implement turn-level partial reward as the average fraction of turns

| | Filtered | Recovered | Combined |
|---|---|---|---|
| *$\tau$2-bench telecom (2,171 non-base tasks)* | | | |
| Trajectories | 158 | 279 | 437 |
| Unrolled examples | 2,049 | 4,814 | 6,863 |
| Recovered:filtered | | 1.77:1 | |
| *BFCL v4 multi-turn (560 train tasks)* | | | |
| base | 96 | 11 | 107 |
| long_context | 78 | 20 | 98 |
| miss_func | 66 | 22 | 88 |
| miss_param | 56 | 36 | 92 |
| Total trajectories | 296 | 89 | 385 |
| Unrolled examples | 1,773 | 954 | 2,727 |
| Recovered:filtered | | 0.30:1 | |

Table 11: Training data composition. Filtered = tasks the teacher passed. Recovered = tasks recovered via per-scenario optimization. For BFCL, the 264 failures (560 train tasks minus 296 passed) yield 89 recoveries, a 33.7% recovery rate.

passed per task:

$$\text{Partial Reward}(t) = \frac{\text{turns passed in task } t}{\text{total turns in task } t}$$

This serves as a per-turn accuracy proxy, analogous to the action accuracy component of $\tau$2-bench's reward decomposition, allowing us to measure whether fine-tuning improves tool-calling behavior at the turn level even when end-to-end task accuracy is unchanged.

## L BFCL Per-Category Results

Table 12 reports per-category accuracy on BFCL v4 multi-turn, averaged across 3 evaluation runs.

| | Gemma 4 E4B-it | | Qwen3-4B-Inst. | |
|---|---|---|---|---|
| **Category** | Base | +Comb. | Base | +Comb. |
| base | 17.2 | 23.3 | 30.0 | 28.9 |
| long_context | 15.0 | 18.3 | 28.3 | 21.1 |
| miss_func | 0.0 | **15.6** | 11.7 | **31.7** |
| miss_param | 17.8 | 21.7 | 24.4 | 20.6 |
| Overall | 12.5 | **19.7** | 23.6 | 25.5 |

Table 12: BFCL per-category accuracy (%, 60 test tasks each, averaged across 3 runs). Bold indicates improvement over baseline.

The `miss_func` category (tasks requiring the model to recognize that a requested function is unavailable) is where both models start lowest (Gemma 0%, Qwen 11.7%) and show the clearest improvement (Gemma 15.6%, Qwen 31.7%). This gain is stable across all three evaluation runs (Appendix M).

BFCL's turn-level evaluation is strict: each turn has ground-truth function calls, and if the model produces no decodable tool calls for a turn, that turn fails immediately regardless of any text response. This makes partial progress harder to observe than in $\tau$2-bench, where the verifier evaluates end-state correctness rather than per-turn output format. Other per-category results show mixed effects at $n$=60 and are exploratory.

| Condition | Run 1 | Run 2 | Run 3 | Spread |
|---|---|---|---|---|
| *Overall accuracy (%, 240 test tasks)* | | | | |
| Gemma baseline | 12.5 | 12.5 | 12.5 | 0.0pp |
| Gemma + combined | 20.4 | 19.6 | 19.2 | 1.2pp |
| Qwen baseline | 23.8 | 23.8 | 23.3 | 0.5pp |
| Qwen + combined | 25.8 | 26.2 | 24.6 | 1.6pp |
| *Per-category max spread (60 test tasks each)* | | | | |
| Gemma baseline | 1.7pp (base, miss_param) | | | |
| Gemma + combined | 3.3pp (long_context) | | | |
| Qwen baseline | 1.7pp (miss_param) | | | |
| Qwen + combined | 3.3pp (base) | | | |

Table 13: BFCL v4 repeated evaluation stability. Overall accuracy is stable across runs (spread $\leq$1.6pp). Per-category results at $n$=60 vary by up to 3.3pp from serving non-determinism alone. The `miss_func` gain is the most stable signal: Gemma 0.0% $\times$3 (baseline) vs. 15.0–16.7% (combined); Qwen 11.7% $\times$3 (baseline) vs. 31.7% $\times$3 (combined).

| Run | Description | Pass^1 |
|---|---|---|
| Original | Train + eval | 0.529 |
| Re-eval | Same checkpoint, new eval | 0.518 |
| Retrain | New training run + eval | 0.538 |

Table 14: Qwen3-4B-Instruct-2507 combined training stability on $\tau$2-bench telecom (114 tasks, 3 trials, temp=0). Pass^1 ranges from 0.518 to 0.538 across independent training and evaluation runs (spread: 0.020), confirming that the reported improvement is not an artifact of a favorable seed.

## M BFCL Repeated Evaluation

To assess serving non-determinism, we ran three independent evaluations per condition on BFCL v4 multi-turn. Despite deterministic user turns and temp=0, vLLM serving introduces small floating-point differences that flip 1–3 tasks per run.

## N Training Stability Verification

To verify that reported results reflect stable training outcomes rather than favorable random seeds, we conducted independent retraining and re-evaluation runs for our strongest result (Qwen3-4B-Instruct-2507 combined on $\tau$2-bench telecom).

The re-evaluation uses the same merged check-

point served under identical vLLM settings; the difference from the original (0.529 vs. 0.518) reflects stochastic user simulation variance. The retrain uses a fresh QLoRA training run with identical hyperparameters; its result (0.538) confirms that the large gain over the baseline (0.132) and filtered-only training (0.111) is stable across independent training and evaluation on the same dataset.