Pareto-Dominant Clarification: Post-Training Coding LLMs via PPO-Lagrangian Budget Constraints

Abhinav Rajput *
Center for Data Science
New York University
New York, NY, USA
abhinav.rajput@nyu.edu

Acey Vogelstein *
Center for Data Science
New York University
New York, NY, USA
av3848@nyu.edu

Abstract

Coding agents operating under ambiguous instructions or user prompts must decide whether to ask clarifying questions or attempt a solution directly. While clarification from the user may improve the correctness of the agent's solution, each back-and-forth interaction incurs user and system costs, forming an explicit accuracy vs. efficiency tradeoff. Existing works study clarification behavior but do not train policies under enforceable clarification budgets; penalty-based approaches typically require separate coefficient tuning swept across all clarification budget levels. We formulate clarification as a Constrained Markov Decision Process (CMDP) and post-train Qwen2.5-Coder-7B-Instruct with PPO-Lagrangian to optimize coding accuracy, subject to an expected question-budget constraint. Evaluated on HumanEvalComm with a GPT-4o-mini oracle simulator, the resulting policies reveal that untuned clarification behavior is Pareto-inefficient: budget-constrained policies can simultaneously achieve higher accuracy and lower clarification rates than the baseline model. Across budget levels, we observe a log-shaped Pareto frontier with diminishing returns to additional clarification. Gains arise not from simply asking more questions overall, but from improved question targeting and better code generation under ambiguity. Without explicit supervision, trained policies learn to allocate clarification budget non-uniformly, asking more frequently on tougher (multi-degradation) tasks. These results suggest that unconstrained interactive LLM systems may systematically use clarification inefficiently.

1 Introduction

Coding agents given ambiguous, incomplete, or inconsistent task specifications must decide at each turn: ask a targeted clarifying question to recover missing information, or submit code directly and risk failure. Asking can be decisive on genuinely underspecified tasks, but each clarification turn costs user attention and system resources. Learning *when* to ask (and, jointly, *what* to ask) under an explicit, enforceable budget on expected clarification cost is the problem we address. Because a single policy generates both the routing decision and the clarifying question, our method optimizes the two together rather than treating question content as fixed.

Prior work on clarifying questions (Rao and Daumé III, 2018; Aliannejadi et al., 2019; Wu and Fard, 2025; Curuksu, 2024) and constrained RL and LLM alignment (Altman, 1999; Stooke et al., 2020; Ouyang et al., 2022) does not train the ask-vs-answer routing policy (the decision at each turn whether to ask a clarifying question or submit code) end-to-end under an explicit per-episode clarification-cost budget. The CMDP framing naturally accommodates arbitrary budgets within a single training run, whereas penalty-based approaches require per-budget coefficient tuning.

We formalize the ask-vs-answer decision as a **Constrained Markov Decision Process** (CMDP; Altman 1999), in which the agent maximizes **pass@1** (defined as the average fraction of unit test cases passed per problem, a continuous score in $[0, 1]$) subject to a budget d_1 on the expected per-episode question count. We post-train **Qwen2.5-Coder-7B-Instruct** (Hui et al., 2024) with LoRA (Hu et al., 2021) via **PPO-Lagrangian** (Schulman et al., 2017; Altman, 1999; Stooke et al., 2020), whose dual-ascent Lagrange multiplier enforces the budget without per-budget penalty tuning. We evaluate policies in $(\bar{q}, \text{pass@1})$ space; a policy **Pareto-dominates** another if it achieves strictly higher pass@1 *and* strictly lower $\bar{q}$ simultaneously. Our contribution is not a new RL algorithm but the end-to-end, budget-constrained post-training of the ask-vs-answer routing policy: the CMDP formulation (Eq. (2)) is what defines

* Equal contribution.

the problem, and other constrained-RL optimizers could target the same formulation.

Contributions.

- We show that untuned LLMs may have Pareto-inefficient clarification policies: PPO-Lagrangian training simultaneously achieves higher accuracy *and* lower question asking rates, correcting the baseline's operating point within a single training paradigm.
- Empirical Pareto analysis across seven budget levels reveals a log-shaped frontier with an efficient operating point: beyond a budget threshold, the friction cost of additional clarifications outweighs their marginal accuracy benefit.
- Two structural findings help explain the gains: policies implicitly learn (1) *when to ask*, allocating their question budget non-uniformly without explicit supervision and asking more on harder multi-degradation problems, and (2) *what to ask*, improving question quality and code generation under ambiguity.

The complete implementation and trained checkpoints are available at our GitHub repository.

2 Related Work

Clarifying questions in NLP. Rao and Daumé III (2018) rank clarification candidates via neural expected value of perfect information on StackExchange data. Aliannejadi et al. (2019) study clarifying questions in open-domain information-seeking dialogue. Wu and Fard (2025) benchmark LLM asking behaviour on degraded coding specifications without training a budget-constrained policy. Curuksu (2024) trains a clarification fallback policy via PPO in multi-turn orchestration environments, but without an explicit per-episode budget constraint; their reward is defined over tool/intent matching rather than task correctness. Recent work studying uncertainty-aware and learned clarification: Zhang and Choi (2025) and Suri et al. (2026) estimate when clarification is useful, while the latter also uses RL, but neither trains under explicit clarification-cost constraints. Sun et al. (2025) uses multi-objective RL with interaction costs incorporated through reward shaping rather than constrained optimization. In coding, Darji and Lutellier (2025) trains clarification components, while Vijayvargiya et al. (2026) evaluates interactive agents under underspecified tasks; neither trains a budget-constrained clarification policy.

Constrained RL and alignment. Constrained MDPs (Altman, 1999) formalise budget-constrained optimisation; PID-Lagrangian dual updates (Stooke et al., 2020) and reward-constrained policy optimisation (Tessler et al., 2018) address stability in constrained RL. Budgeted and cost-aware dialogue policies have likewise been framed as CMDPs (Zhang et al., 2019; Waki et al., 2025); our contribution is not the CMDP framing itself but its application to end-to-end, budget-constrained post-training of the ask-vs-answer routing policy for multi-turn coding agents. Reinforcement Learning from Human Feedback (RLHF; Ouyang et al. 2022) aligns LLMs via human feedback without per-episode cost constraints; Safe RLHF (Dai et al., 2023) extends this via a CMDP formulation with PPO-Lagrangian to trade off helpfulness against a harmlessness cost.

3 Method

3.1 CMDP Formulation

We model the multi-turn clarification task as a CMDP (Altman, 1999) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, C_q, C_t, d_1, d_2)$. Full formal definitions (state, transition function, reward) are in Appendix A.

Actions.

$$a_t \in \{\texttt{[ASK]}\ q_t, \quad \texttt{[ANSWER]}\ c_t\}, \tag{1}$$

where q_t is a clarifying question and c_t is Python code; routing is via prefix log-probability scoring (Section 3.2).

Costs and budget. The primary cost $C_q(a_t)$ counts the number of distinct atomic questions in an [ASK] turn, as determined by the user simulator (details in Appendix A); $C_q = 0$ for [ANSWER]. Counting individual questions rather than turns prevents the agent from exploiting the budget by bundling multiple requests into a single [ASK]. The secondary cost $C_t(a_t) = 1$ accrues per turn. Budget d_1 constrains the question count per episode on average across the training distribution; $d_2 = 4$ provides a soft turn-count constraint below the hard cap $T_{\max} = 6$.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_\pi[R] \text{ s.t. } \mathbb{E}_\pi[C_i] \le d_i,\ i \in \{q, t\}. \tag{2}$$

3.2 Interaction Protocol

At each turn the agent outputs `[ASK]` (passed to a GPT-4o-mini simulator, which returns a factual answer) or `[ANSWER]` (code executed against hidden tests; episode terminates). Termination conditions and routing details are in Appendix A.

3.3 PPO-Lagrangian Optimization

Policy and value heads. The policy is Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) with LoRA (Hu et al., 2021) (r=16, α=32); architecture, value head design, and training objective details are in Appendix A. We solve Eq. (2) via Lagrangian relaxation (Altman, 1999; Stooke et al., 2020), approximated via a simultaneous primal-dual method over policy parameters and non-negative Lagrange multipliers λ_1 (question budget) and λ_2 (turn budget), with $\lambda_1, \lambda_2 \geq 0$.

Generalized Advantage Estimation (Schulman et al., 2016). Three GAE streams for $X \in \{R, C_q, C_t\}$ (recurrence in Appendix A) are whitened to zero mean and unit variance before combination:

$$A_t^{\text{lag}} = \tilde{A}_t^R - \lambda_1 \tilde{A}_t^{C_q} - \lambda_2 \tilde{A}_t^{C_t}. \quad (3)$$

The policy is updated via the clipped PPO objective (Schulman et al., 2017) (ε=0.2) over continuation tokens only, with prefix tokens excluded from likelihood ratios (equation in Appendix A).

Dual update. After each rollout batch, and *before* the PPO gradient step:

$$\lambda_1 \leftarrow \text{clip}\big(\lambda_1 + \eta_{\lambda_1}(\bar{q} - d_1),\ 0,\ \lambda_{1,\max}\big), \quad (4)$$
$$\lambda_2 \leftarrow \text{clip}\big(\lambda_2 + \eta_{\lambda_2}(\bar{C}_t - d_2),\ 0,\ \lambda_{2,\max}\big), \quad (5)$$

where $\bar{C}_t$ is the mean turn count per rollout batch; η_{λ_1}=0.1, η_{λ_2}=0.005; λ bounds in Table 1. Training setup, checkpoint selection criteria, and Pareto frontier construction are detailed in Appendix A.

4 Experimental Setup

Benchmark and evaluation metric. We train and evaluate on HumanEvalComm (Wu and Fard, 2025), a multi-turn extension of HumanEval (Chen et al., 2021) in which each problem specification has been deliberately degraded before being presented to the model.[1] Each episode presents the model with a degraded specification; the model may ask clarifying questions (answered by a GPT-4o-mini simulator from the ground-truth oracle) before submitting a Python solution. Every instance presented to the model is degraded; HumanEvalComm contains no unmodified specifications. Our primary evaluation metric is pass@1 (defined in Section 1). We split 772 degraded instances into 303 for training, 52 for validation, and 417 for held-out testing (split details in Appendix A).

Policy and simulator. We fine-tune Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) using LoRA (Hu et al., 2021)[2]; both the policy and simulator receive a system prompt (full text and rationale in Appendix E; representative episodes in Appendix D). We train 10 policies at seven budget levels, with two independent runs (different seeds, otherwise identical) at $d_1 \in \{0.5, 0.75, 1.0\}$ to measure run-to-run variance. The baseline is the frozen, untrained model (mean pass@1 = 0.690, $\bar{q}$ = 0.856). Notably, the budget d_1 is enforced only during training via the Lagrange multiplier; at evaluation the policy generates freely, and budget compliance is assessed post-hoc from the resulting question rates. Checkpoint selection and training curves are in Appendix A.

5 Results

Budget compliance and the Pareto frontier. Figure 1 plots all trained policies in ($\bar{q}$, pass@1) space. The key efficiency result is d_1=0.5: at $\bar{q}$ = 0.453 (roughly half the baseline's question rate of 0.856), two independent runs achieve mean pass@1 $\in \{0.683, 0.719\}$, neither significantly different from baseline (both $p \geq 0.10$, paired t-test), confirming that the budget constraint is enforceable at half the question rate without degrading mean pass@1. Allowing more questions continues to help: d_1=0.75 ($\bar{q}$ = 0.782) achieves mean pass@1 = 0.761 (+7.1%, best run; other run +5.8%, both p<0.01), and d_1=1.0 ($\bar{q}$ = 0.859) achieves mean pass@1 = 0.778 (+8.7%, best run; other run +7.4%, both p<0.0001). At d_1=1.0, λ_1 activates at some iterations but does not **bind** persistently; above d_1=1.0, $\lambda_1 \approx 0$ throughout training (the budget is slack, see Appendix B), and

[1] Seven degradation types: 1a, 1c, 1p, 2ac, 2ap, 2cp, 3acp; numeric prefix = number of simultaneous degradations; letter suffix = type (**a** = ambiguity, **c** = contradiction/inconsistency, **p** = partial truncation).

[2] Artifact licenses: HumanEval is released under the MIT License; Qwen2.5-Coder-7B-Instruct and HumanEvalComm are released under Apache-2.0. The GPT-4o-mini simulator is accessed via the OpenAI API under its terms of use. All artifacts are used in a manner consistent with their intended research use.

further budget increases yield no marginal accuracy gain. In this slack regime, PPO trains without active question-budget pressure yet still improves accuracy over the baseline; these high-budget runs ($d_1 \in \{1.5, 2.0\}$, $\lambda_1 \approx 0$) therefore serve as an approximate unconstrained-PPO control, indicating that PPO itself contributes to the accuracy gains, while the Lagrangian constraint is what provides budget-targeted control and yields the efficient operating point. Across all runs, the turn multiplier λ_2 remained 0 at every dual update: the maximum batch-mean turn count was 3.33, below the turn budget d_2=4. The trained policies are therefore behaviorally question-only, so our main claims rest on the question constraint rather than the turn constraint; d_2 serves only as a safeguard against pathological dialogues below the hard cap $T_{\max}$=6.

Diminishing returns and the value of clarification. The Pareto frontier in Figure 1 follows a log-shaped curve: the marginal pass@1 gain per additional clarifying question decreases as the budget grows. The step from d_1=0.5 to d_1=0.75 yields a substantially larger accuracy gain than the step from d_1=0.75 to d_1=1.0, and beyond d_1=1.0 there is no further marginal accuracy gain. This implies that, beyond a certain budget, the friction cost of each additional clarification outweighs its marginal accuracy benefit.

Among our trained policies, d_1=0.75 is the only one to **Pareto-dominate** the baseline: it achieves significantly higher pass@1 (+5.8–+7.1%, both $p<0.01$) while asking significantly fewer questions ($\bar{q} \in \{0.719, 0.782\}$ vs. 0.856; both $p<0.01$, paired t-test) across both independent runs. The d_1=0.5 policy achieves comparable pass@1 at roughly half the baseline question rate (both $p \geq 0.10$); one run is above baseline and one below, consistent with budget compliance but not a reliable accuracy gain at this budget level. Appendix D (Episode 2) illustrates Pareto domination concretely. Policies at $d_1 \in \{0, 0.25\}$ fall below baseline (details in Appendix A), confirming that suppressing clarification too aggressively is counterproductive.

Performance by degradation complexity. Figure 2 shows pass@1 broken down by degradation complexity group: **1-deg** (types 1a, 1c, 1p; n=266) and **2- or 3-deg** (types 2ac, 2ap, 2cp, 3acp; n=151). Every policy scores lower on harder multi-degradation problems, where a single clarifying question can resolve at most one simultaneous

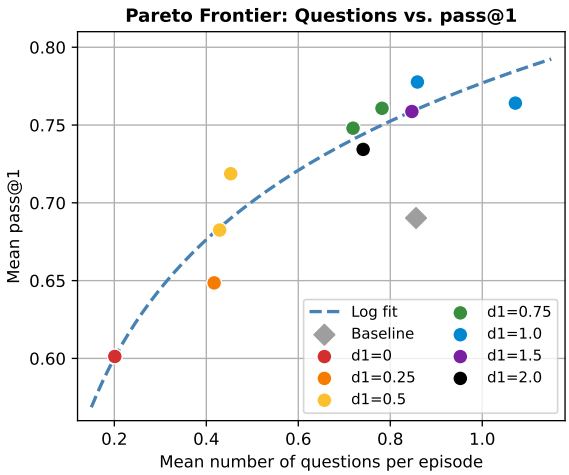

Figure 1: Pareto frontier in ($\bar{q}$, pass@1) space for all trained policies and the untrained baseline (gray diamond), evaluated on 417 held-out problems under greedy decoding. The log fit illustrates diminishing returns; from d_1=1.0 onward Lagrangian enforcement weakens or ceases.

spec defect; trained policies tend to ask more questions on these cases. Question-count breakdowns by group and by all seven degradation types are in Appendix C.

Quality over quantity: better questions and code. Controlling for question count (1Q→1Q episodes, i.e., episodes where both the baseline and the trained policy asked exactly one question), d_1=0.75 achieves +7.1% over baseline ($p<0.001$, n=490 episodes pooled across both runs, paired t-test), showing gains do not stem from higher question counts. The per-episode breakdown (Appendix C) is consistent with better question targeting and code quality under ambiguity.

6 Conclusion

CMDP post-training with PPO-Lagrangian produces a Pareto frontier in $(\bar{q}, \text{pass@1})$ space: trained policies simultaneously achieve higher accuracy and lower question rates than the untuned baseline. Two structural findings extend this result. First, policies where the Lagrangian is actively enforced ($d_1 \in \{0.5, 0.75\}$, $\lambda_1 > 0$) ask more questions on harder multi-degradation problems while asking fewer questions overall than the baseline (Figure 2 and Appendix C), suggesting that CMDP training reallocates question budget toward more informative contexts rather than reducing it. Second, gains are driven by question quality and code generation ability rather than question count: trained policies significantly outperform the baseline even

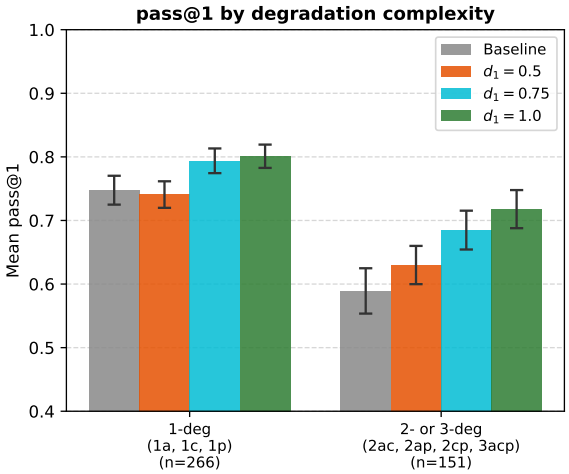

Figure 2: Mean pass@1 by degradation type (417 problems, greedy decoding). $d_1 \in \{0.5, 0.75, 1.0\}$ results averaged across two independent runs. Error bars are ± 1 standard error of the mean (SEM).

when both ask exactly one question.

The Pareto-dominant operating point (d_1=0.75, within the binding regime where λ_1>0) demonstrates that active Lagrangian enforcement achieves simultaneous quality and efficiency gains. The log-shaped Pareto frontier further suggests a practical design principle: there is a budget threshold beyond which the friction cost of additional clarifications outweighs their marginal accuracy benefit, and operating beyond it yields no gain. Future work should examine this framework on naturally occurring ambiguity, larger model scales, and human rather than simulator oracles. A natural extension is a *dynamic* budget that adapts to the estimated complexity or ambiguity of each problem, rather than enforcing a fixed expected question count across all episodes.

Limitations

Run-to-run variance. Independent training runs at the same budget ($d_1 \in \{0.5, 0.75, 1.0\}$) converge to notably different mean pass@1 values (e.g., 0.683 vs. 0.719 at d_1=0.5; 0.778 vs. 0.764 at d_1=1.0), indicating sensitivity to random initialisation. This limits the reliability of conclusions drawn from any single run and motivates future work on more stable training procedures.

Training scale and instability. All policies were trained under significant compute constraints: only 80 iterations with a mini-batch size of 8 episodes and 32 episodes per rollout iteration. Small batches produce high-variance gradient estimates, which destabilises the dual-ascent updates for the Lagrange multipliers and likely contributes to the run-to-run variance noted above. Larger batch sizes, more iterations, and learning-rate schedules tuned for the Lagrangian dynamics would likely improve both convergence reliability and final performance.

LLM-based question simulator. The oracle that answers clarifying questions is GPT-4o-mini running at temperature 0, making its responses deterministic given the question and ground-truth specification. This eliminates answer inconsistency as a noise source, but a greedy deterministic simulator may not reflect the variability of real human responses: a human might paraphrase, hedge, or volunteer adjacent information differently across interactions, all of which would affect the policy's learned question-asking strategy.

Single benchmark and small training set. All results are on HumanEvalComm, a single code-generation benchmark with a specific set of synthetic degradation types. Whether the learned strategies and efficiency gains generalise to other task types or to naturally occurring ambiguity is untested. The training pool is also small: 303 degraded instances from 64 base problems, smaller than the 417-problem held-out test set. Whether this scale is sufficient for reliable convergence is unclear, and likely contributes to the run-to-run variance observed across seeds.

Scale. Experiments use a 7B-parameter model with LoRA fine-tuning. Whether the CMDP approach scales to larger models, whether the emergent budget-allocation behaviour persists, and whether the Lagrange multiplier dynamics remain stable at larger scale are open questions.

Discrete budget coverage. Due to compute constraints, we train at only seven discrete budget values ($d_1 \in \{0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0\}$); the log fit in Figure 1 interpolates between them. The true shape of the Pareto frontier between trained budgets is unverified; the optimal trade-off curve could differ meaningfully from the fitted log curve in these intervals.

Disentangling question quality from code quality. PPO training jointly optimises question-asking and code generation, making it difficult to attribute gains solely to improved clarification behaviour. Cases where a policy asks comparable questions to the baseline yet produces better code suggest that general coding ability also improves

under PPO. Future work should design controlled ablations, for instance freezing the code-generation head and training only the question-asking policy, to cleanly separate these two sources of improvement.

Acknowledgments

The authors used Claude (Anthropic) for assistance with writing and editing during this project. All experimental design, analysis, implementation, math, and final verification were performed by the authors.

References

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. 2019. Asking clarifying questions in open-domain information-seeking conversations. *CoRR*, abs/1907.06554.

Eitan Altman. 1999. *Constrained Markov Decision Processes*. Chapman & Hall/CRC, Boca Raton, FL.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Jeremy Curuksu. 2024. Policy optimization to align the fidelity and efficiency of reasoning agents in multi-turn dialogues. In *NeurIPS 2024 Workshop on Open-World Agents*.

Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang. 2023. Safe rlhf: Safe reinforcement learning from human feedback. *Preprint*, arXiv:2310.12773.

Harsh Darji and Thibaud Lutellier. 2025. Curiosity by design: An llm-based coding assistant asking clarification questions. *Preprint*, arXiv:2507.21285.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit optimizers via block-wise quantization. *Preprint*, arXiv:2110.02861.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Sudha Rao and Hal Daumé III. 2018. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2737–2746, Melbourne, Australia. Association for Computational Linguistics.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Preprint*, arXiv:1707.06347.

Adam Stooke, Joshua Achiam, and Pieter Abbeel. 2020. Responsive safety in reinforcement learning by PID lagrangian methods. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9133–9143. PMLR.

Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. 2025. Training proactive and personalized llm agents. *Preprint*, arXiv:2511.02208.

Manan Suri, Puneet Mathur, Nedim Lipka, Franck Dernoncourt, Ryan A. Rossi, and Dinesh Manocha. 2026. Structured uncertainty guided clarification for LLM agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 40811–40838, San Diego, California, United States. Association for Computational Linguistics.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. 2018. Reward constrained policy optimization. *Preprint*, arXiv:1805.11074.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. 2026. AmbigSWE: Interactive agents to overcome underspecificity in software engineering. In *The Fourteenth International Conference on Learning Representations*.

Issei Waki, Ryu Takeda, and Kazunori Komatani. 2025. Learning to ask efficiently in dialogue: Reinforcement learning extensions for stream-based active learning. In *Proceedings of the 26th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 431–440, Avignon, France. Association for Computational Linguistics.

Jie JW Wu and Fatemeh H. Fard. 2025. Humanevalcomm: Benchmarking the communication competence of code generation for llms and llm agent. *ACM Transactions on Software Engineering and Methodology*.

Michael JQ Zhang and Eunsol Choi. 2025. Clarify when necessary: Resolving ambiguity through interaction with LMs. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5541–5558, Albuquerque, New Mexico. Association for Computational Linguistics.

Zhirui Zhang, Xiujun Li, Jianfeng Gao, and Enhong Chen. 2019. Budgeted policy learning for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3742–3751, Florence, Italy. Association for Computational Linguistics.

Appendix

A Implementation Details

Table 1 consolidates all hyperparameters used across all training runs. All policies use identical hyperparameters; only the budget d_1 varies.

CMDP formal definitions. **State.** At turn t, s_t is the full conversation history: the degraded problem specification x together with all prior question-answer pairs $\{(q_1, u_1), \ldots, (q_{t-1}, u_{t-1})\}$. **Transition function.** For `[ANSWER]` actions, the episode terminates deterministically. For `[ASK]` actions, s_{t+1} incorporates the GPT-4o-mini user simulator's response to q_t; the simulator runs at temperature $= 0$, making its responses deterministic given the question and ground-truth specification. **Reward.** $R(s_t, a_t) = \rho(c_t)$ if $a_t =$ `[ANSWER]` c_t, else 0, where $\rho(c_t) \in [0, 1]$ is pass@1: the fraction of hidden test assertions passed. Reward is non-zero only at the terminal step. The training signal optimizes pass@1.

Policy and value heads. We parameterize the policy as Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) with LoRA (Hu et al., 2021) ($r = 16$, $\alpha = 32$) applied to all seven attention and MLP projection layers (≈40M of 7B parameters trainable). Three value heads V^R, V^{C_q}, V^{C_t}, each implemented as $\text{Linear}(h, 1024) \to \text{GELU} \to \text{LayerNorm}(1024) \to \text{Linear}(1024, 1)$ over the final-layer last-prompt-token hidden state, estimate expected future returns for the reward, question-cost, and turn-cost streams. Output layers are zero-initialized; value MSE gradients backpropagate through the shared LoRA representation. A frozen reference copy (base weights, no LoRA) runs on a second GPU for KL regularization; a rollout copy (LoRA weights synced each iteration) handles episode collection. Both π_θ and $\pi_{\theta_{\text{old}}}$ are computed via clean forward passes, bypassing HuggingFace logit processors that would otherwise corrupt log-probability scales.

Component	Hyperparameter	Value
Policy (LoRA)	Rank r	16
	Scale α	32
	Trainable parameters	≈40M
	Learning rate	5×10^{-6}
	LR warmup	20 steps (linear)
Value heads	Hidden dim	1024
	Activation	GELU
	Learning rate	10^{-4}
	Output layer init	zeros
PPO	Clip ε	0.2
	Mini-batch size	8
	Max epochs / iteration	4
	KL early-stop threshold	0.05
	β_{KL}	0.25
	β_H	0.01
	Value loss coefficient	0.5
GAE	γ	1.0
	λ_{GAE}	0.95
Lagrangian	η_{λ_1}	0.1
	η_{λ_2}	0.005
	$\lambda_{1,\max}$	10
	$\lambda_{2,\max}$	5
	Turn budget d_2	4
Training	Episodes / iteration	32
	Total iterations	80
	Checkpoint frequency	every 5 iterations
	Rollout temperature	0.8
	Eval temperature	0 (greedy)
Episode	Max turns $T_{\max}$	6
	Optimizer	8-bit AdamW
Gradient clipping	ℓ_2 norm	1.0

Table 1: Hyperparameters used for all training runs.

PPO surrogate and prefix exclusion. The policy is updated via the clipped PPO objective (Schulman et al., 2017) ($\varepsilon = 0.2$):

$$L^{\text{PPO}} = -\mathbb{E}_t\Big[\min\Big(r_t \hat{A}_t^{\text{lag}}, \text{clip}(r_t,\, 1-\varepsilon,\, 1+\varepsilon)\, \hat{A}_t^{\text{lag}}\Big)\Big]. \tag{6}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_{\text{cont},t} \mid s_t, \text{pref}_t)}{\pi_{\theta_{\text{old}}}(a_{\text{cont},t} \mid s_t, \text{pref}_t)}$$

is the probability ratio over *continuation tokens only*; the forced `[ASK]`/`[ANSWER]` prefix is excluded from both numerator and denominator.

Routing gradient. Routing and generation share the same LoRA weights: the prefix scores used to select `[ASK]` vs. `[ANSWER]` at rollout time are computed from the same attention and MLP projections that PPO updates via continuation-token gradients. When λ_1 rises, the $-\lambda_1 \tilde{A}_t^{C_q}$ term penalises `[ASK]`-turn continuations; the resulting weight updates shift the model's logits at the routing step in the next iteration. Budget enforcement thus propagates to routing through shared parameters across training iterations, which is the standard mechanism by which PPO shapes structured generation behaviour. As a concrete two-token example, suppose the policy emits `[ASK] what type?`. The `[ASK]` prefix tokens are excluded from the ratio r_t, so the gradient never directly rewards or penalises the *decision* to ask; only the continuation `what type?` enters the loss. Yet because the prefix logits and the continuation are produced by the same shared LoRA weights, updating those weights to make a useful `what type?` more likely also perturbs the logits that produced `[ASK]`: the routing decision is coupled to the continuation gradient and is shaped as a side effect of it, rather than being fixed. The routing decision is therefore learned indirectly, through the shared parameters, rather than optimised directly against the sparse pass@1 signal, which would otherwise attribute the entire episode's outcome to a single decision token.

Episode protocol. Each episode begins with the agent receiving a degraded problem specification. At each turn: (1) the agent generates an action that begins with `[ASK]` or `[ANSWER]`, selected via prefix-scored routing; (2) **if `[ASK]`**, the question is passed to the GPT-4o-mini user simulator, which holds the complete specification and returns a factual answer together with an atomic question count N, setting $C_q = N$; (3) **if `[ANSWER]`**, code is executed against hidden tests, yielding $\rho = k/n$ (k of n assertions passing), and the episode terminates. Episodes reaching $T_{\max} = 6$ terminate with $R = 0$.

Atomic question counting. The simulator appends `QUESTION_COUNT: N` to its reply, where N is the number of distinct information requests in the `[ASK]` turn. Multi-question turns are charged $C_q = N > 1$, preventing budget exploitation by bundling questions.

Prefix-scored routing. Given s_t, the model performs a forward pass over the prompt and scores each prefix via log-softmax over the final-position logits. Under the Qwen2.5-Coder tokenizer, `[ASK]` and `[ANSWER]` each tokenize as four tokens but share the first (`[`), so the model performs a second forward step via KV cache to score the second distinguishing token of each prefix (`ASK` vs. `ANSWER`). The prefix with the higher two-token sum of log-softmax values is selected; this is a heuristic approximation to the full four-token joint log-probability and is distinct from hard-constraint decoding algorithms that guarantee structural constraints (tries, finite-state acceptors, grammar-constrained decoding). At evaluation (temperature $= 0$), the higher-scoring prefix is selected greedily; during training rollouts (temperature $= 0.8$), the prefix is sampled from a Boltzmann distribution over the two scores. Ties at greedy decoding default to `[ASK]`. The model generates freely after the chosen prefix.

GAE recurrence (Schulman et al., 2016). Three advantage streams are computed via GAE for $X \in \{R, C_q, C_t\}$:

$$\begin{aligned} \delta_t^X &= x_t + \gamma V^X(s_{t+1}) - V^X(s_t), \\ A_t^X &= \sum_{l \geq 0} (\gamma \lambda_{\text{GAE}})^l \, \delta_{t+l}^X. \end{aligned} \tag{7}$$

where x_t is the stream-specific signal at turn t: $R(s_t, a_t)$ for $X{=}R$, $C_q(a_t)$ for $X{=}C_q$, and $C_t(a_t)$ for $X{=}C_t$. The $\gamma{=}1.0$, $\lambda_{\text{GAE}}{=}0.95$ setting introduces an effective per-turn discount of 0.95^τ on advantages at turn depth τ from the episode end.

Double whitening. Let $\hat{A}_t^{\text{lag}} = \text{whiten}(A_t^{\text{lag}})$ denote the result of whitening A_t^{lag} once more before the PPO surrogate for numerical stability; this second pass scales all terms by $1/\sqrt{1 + \lambda_1^2 + \lambda_2^2}$ (under stream independence), so λ_1 shifts the advantage threshold rather than amplifying the absolute gradient magnitude of the constraint term.

Full training objective.

$$L(\theta) = L^{\text{PPO}} + \beta_{\text{KL}} L_{\text{KL}} - \beta_H L_H + 0.5\, L^{\text{vf}}, \tag{8}$$

where $L_{\text{KL}} = \max(0, \mathbb{E}[\log \pi_\theta - \log \pi_{\text{ref}}])$ is a sequence-level KL from the frozen reference (clamped non-negative, $\beta_{\text{KL}} = 0.25$); $L_H = -\mathbb{E}[\log \pi_\theta / |a_{\text{cont}}|]$ a length-normalized entropy bonus ($\beta_H = 0.01$); and L^{vf} total value MSE across all three heads. All terms are over continuation tokens only. Approx-KL is $\hat{D}_{\text{KL}} = \mathbb{E}[(r_t - 1) - \log r_t]$; if $\hat{D}_{\text{KL}} > 0.05$ per mini-batch, remaining epochs for that iteration are skipped. Gradient ℓ_2 norm is clipped to 1.0.

Dual update mechanics. The λ values entering $\hat{A}^{\text{lag}}$ (via Eq. (3)) within the PPO update are therefore the post-dual-update multipliers of the current iteration. When $\bar{q} > d_1$, λ_1 rises, shifting the sign threshold in $\hat{A}^{\text{lag}}$: some `[ASK]` continuations that ranked as positive flip to negative, making them less likely to be reinforced by PPO. The absolute gradient scale is fixed by the second whitening and is not affected by λ_1.

Per-iteration training steps. Each PPO-Lagrangian iteration proceeds as: (1) sync LoRA weights to the rollout model; (2) sample a batch of 32 episodes from the training pool (user-simulator API calls concurrent via `asyncio`); (3) compute GAE for all three streams; (4) update λ_1, λ_2 via dual ascent (Eqs. (4)–(5)); (5) run up to 4 PPO epochs with early-exit on approx-KL; (6) step the LR scheduler. Policy and value heads use 8-bit AdamW (Dettmers et al., 2022) (LR 5×10^{-6} and 10^{-4}; 20-step linear warmup for the policy, constant thereafter).

Train/eval split. All degraded variants of the same base HumanEval problem are kept in the same split, preventing the model from seeing the ground-truth specification of an eval problem in a different degraded form during training. A fixed set of 52 validation problems (stratified by degradation type, sampled with a fixed seed) is used for all policies, leaving 417 held-out problems for the main eval results reported here.

Training setup. Training uses 303 degraded instances from the 64 base problems not held out for evaluation (not all degradation types are available for every base problem; rare types such as `3acp` exist for only 12 of 164 base problems), running 80 PPO-Lagrangian iterations per budget (≈11h on 2×A100-40GB; hyperparameters in Table 1). The reward signal is exclusively pass@1.

Checkpoint selection. Checkpoints are saved every 5 iterations; the 52-problem validation set (stratified by degradation type, seed 99) is evaluated under greedy decoding at every 10th checkpoint (iter 9, 19, …, 79). A checkpoint *satisfies* the budget if $\bar{q}^{\text{val}} \le d_1$ on the val set under greedy decoding; the primary criterion selects the highest val pass@1 among satisfying checkpoints. If no satisfying checkpoint exists (as at $d_1 = 0$), the fallback selects the checkpoint with the lowest $\bar{q}^{\text{val}}$; the $d_1 = 0$ result therefore reflects training *toward* zero clarification, not a converged zero-question policy. The $d_1 = 0.25$ run is excluded from the frontier: it satisfied the budget at validation but violated it at test time ($\bar{q}^{\text{test}} = 0.417 > 0.25$).

Pareto frontier and evaluation. All seven budget levels are studied; all runs use identical hyperparameters, varying only d_1. All policies are evaluated on the 417-problem held-out test set (Section 4); statistical comparisons use paired t-tests per run. The baseline is untuned Qwen2.5-Coder-7B-Instruct under identical multi-turn conditions (prefix-scored routing, $T_{\max} = 6$, greedy decoding). Budget compliance is assessed per run on the test set via one-sample t-test (H_0: $\mu_q \le d_1$, α=0.05). At d_1=0.75, neither run significantly exceeds the budget (run 1: $\bar{q}$=0.782, p=0.15; run 2: $\bar{q}$=0.719, p=0.86); both are statistically budget-compliant (n=417, SE ≈0.03). At d_1=1.0, run 1 satisfies the budget ($\bar{q}$=0.859, well below budget), while run 2 significantly exceeds it ($\bar{q}$=1.072, p=0.027). The $d_1 = 0$ policy falls below the baseline and is dominated by $d_1 = 0.5$; it does not Pareto-dominate the baseline. The reported frontier ($d_1 \in \{0.5, 0.75, 1.0, 1.5, 2.0\}$) is anchored to the budget-satisfying run at each level; gains from budget-violating runs are reported separately for completeness.

$d_1 = 0.25$ run details. The single run reached $\bar{q}^{\text{val}} = 0.25$ at iter_0079 (val pass@1 $= 0.481$, budget-feasible at val selection), but test-time $\bar{q} = 0.417 > 0.25$; it is excluded from the Pareto frontier because $\bar{q}^{\text{test}} > d_1$.

B Training Curves

Figures 3–5 show training dynamics for representative policies at $d_1 \in \{0.5, 0.75, 1.0\}$. These three budgets span the efficient region of the frontier: d_1=0 never achieves budget compliance within 80 iterations and the single d_1=0.25 run violates the budget at test time, budgets strictly above $d_1 = 1.0$ show $\lambda_1 = 0$ throughout training (the constraint never binds), and $\{0.5, 0.75, 1.0\}$ are the only budgets trained with two independent runs. Each figure shows four panels across the 80 training iterations: (1) mean episode reward, (2) mid-training validation pass@1 evaluated every 10 iterations with the selected checkpoint marked, (3) mean questions per episode $\bar{q}$ with the budget d_1 as a dashed reference line, and (4) the Lagrange multiplier λ_1.

The λ_1 panels directly confirm the budget compliance story from Section 5: for $d_1 = 1.0$, λ_1

activates at some iterations but does not persist throughout training, while for $d_1 \in \{0.5, 0.75\}$, λ_1 rises early and stabilises at a persistently positive value, reflecting the dual-ascent feedback loop enforcing the budget.

C Degradation Type Breakdown

At d_1=1.0, mean pass@1 rises on zero-question episodes (73.7% baseline $\rightarrow$ 81.1%) and on one-question episodes (67.8% $\rightarrow$ 77.4%), consistent with the model learning to write better code under ambiguity and to formulate more targeted questions when clarification is warranted.

Figures 7 and 8 show pass@1 broken down by all seven individual degradation types (rows) and question count: 0 questions, exactly 1 question, and 2+ questions (columns). For each policy, episodes are partitioned per run by that run's question count and pooled across both runs before computing means. This finer-grained view reveals heterogeneous behaviour within the 1-deg group that is obscured in the aggregate Figures 2 and 6: ambiguity problems (1a) are near-ceiling for all models regardless of question count; contradiction problems (1c) show consistent gains for trained policies in both the 0Q and 1+Q columns, reflecting improved code-generation quality for contradictory specs; and truncation problems (1p) show that the 0Q performance gap between trained policies and the baseline is largely a selection effect i.e. trained policies have learned to skip asking on the easier truncation cases, while the baseline skips asking at random.

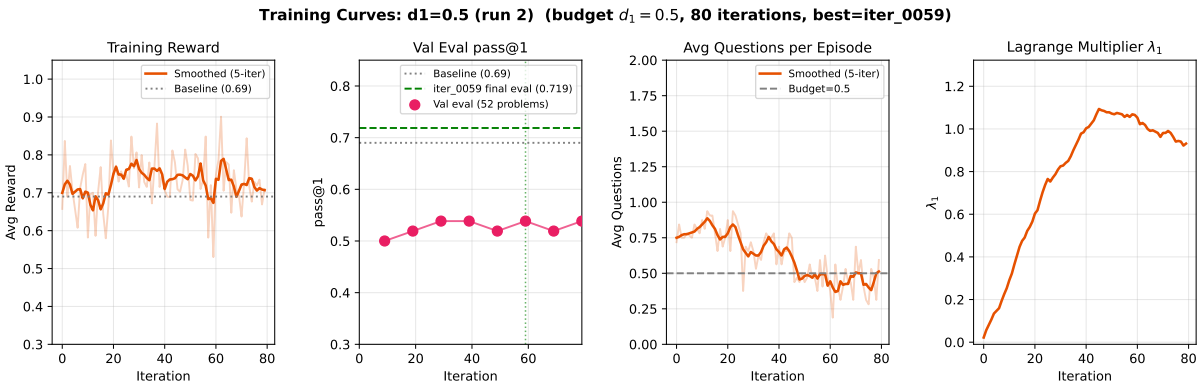

Figure 3: Training curves for $d_1 = 0.5$.

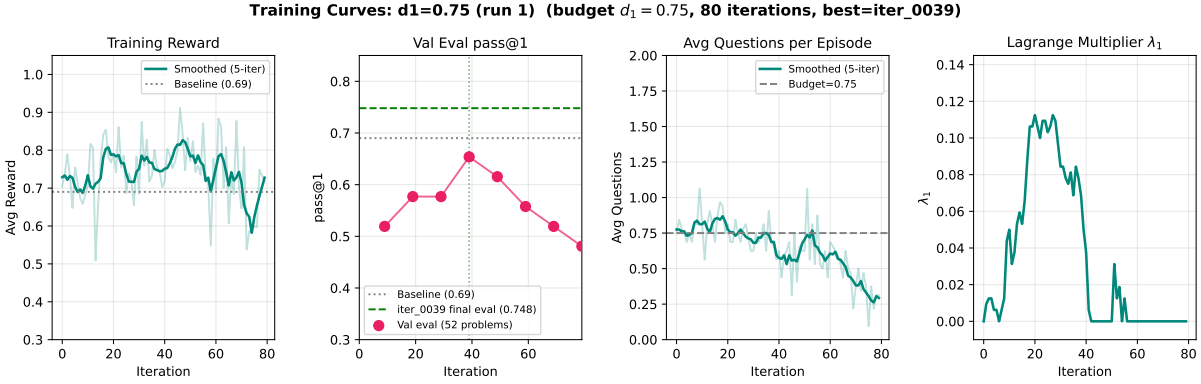

Figure 4: Training curves for $d_1 = 0.75$.

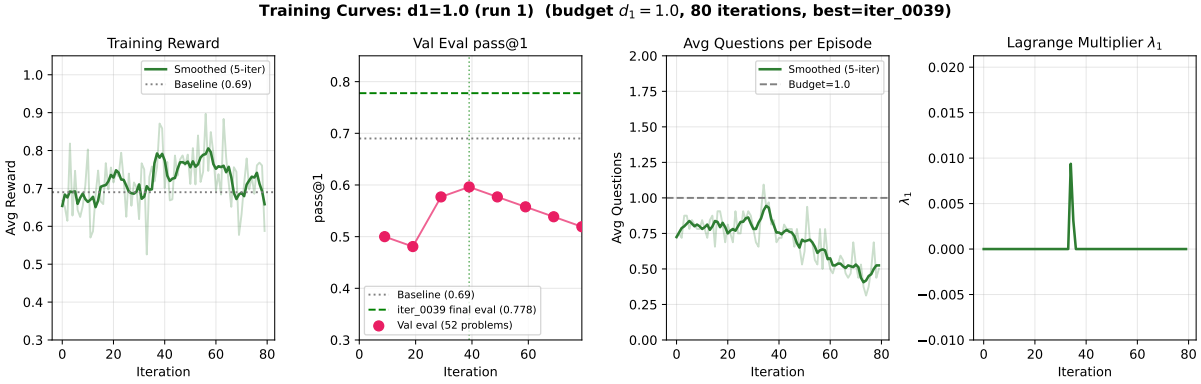

Figure 5: Training curves for $d_1 = 1.0$.

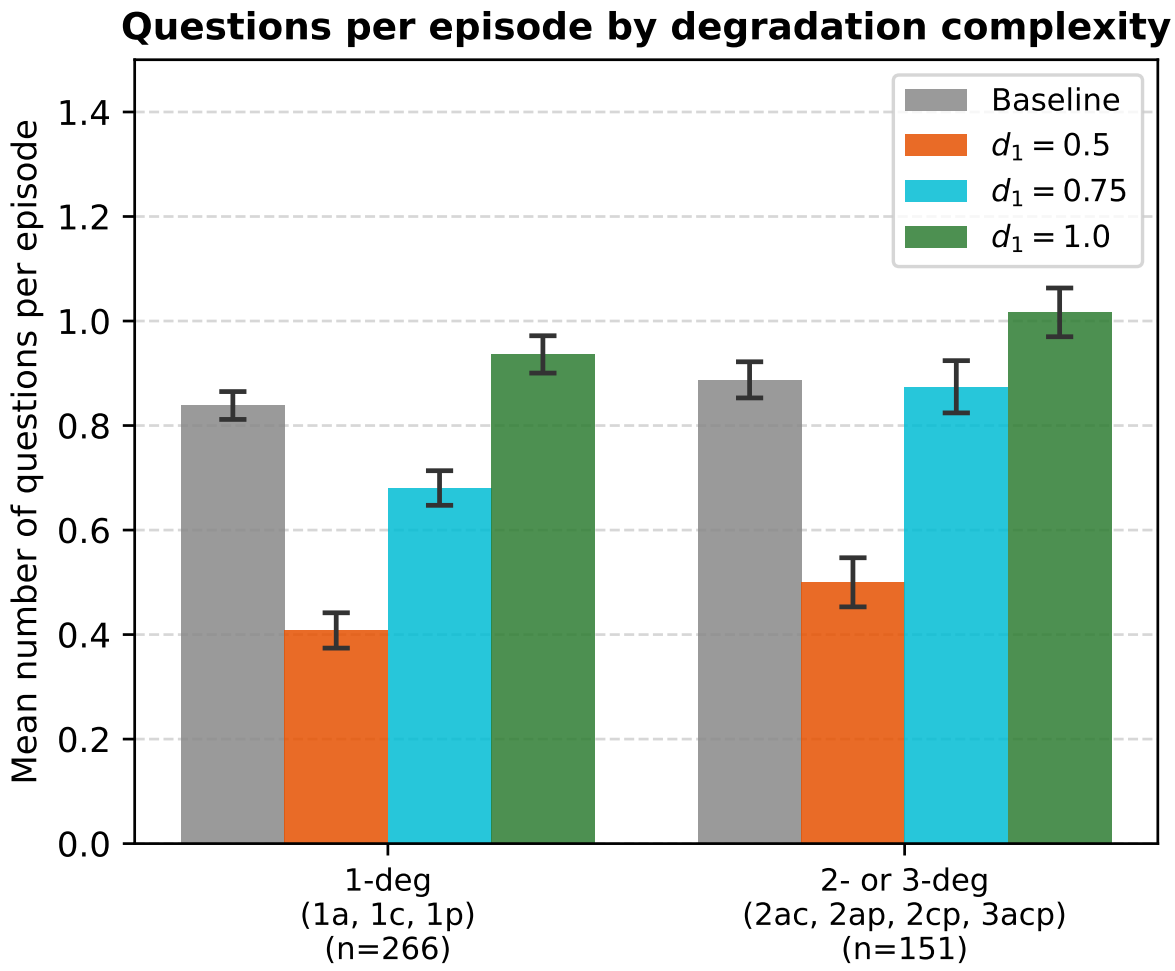

Figure 6: Mean number of questions per episode by degradation complexity group, evaluated on the 417-problem held-out set under greedy decoding. Settings match Figure 2.

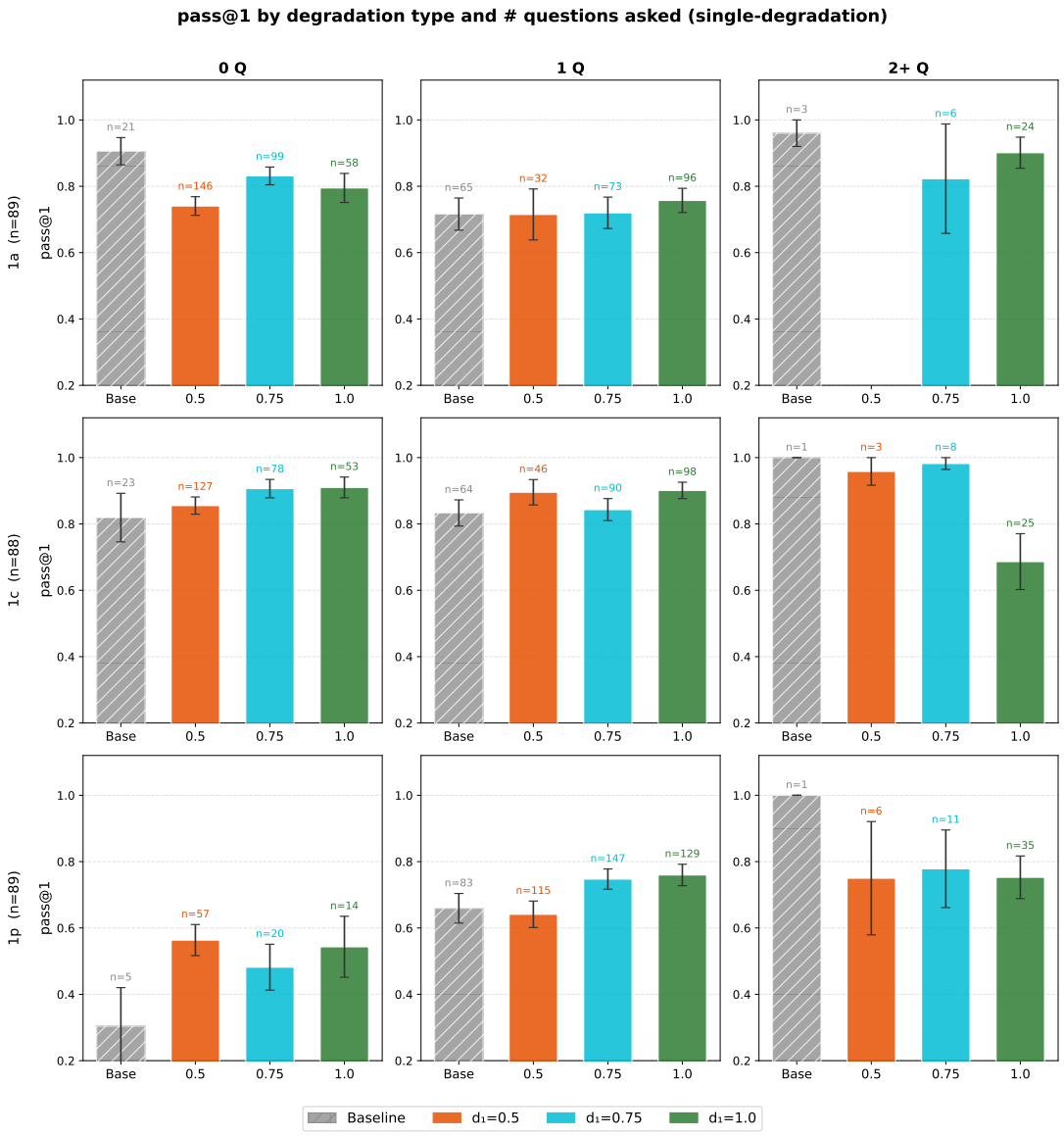

Figure 7: pass@1 by degradation type and number of questions asked, for single-degradation types (1a, 1c, 1p). Columns correspond to episodes where 0, exactly 1, or 2+ clarifying questions were asked. Per-run pooling: for policies with two independent runs, each run contributes its episodes independently, classified by that run's question count, then pooled before computing means. Error bars are ± 1 SEM.

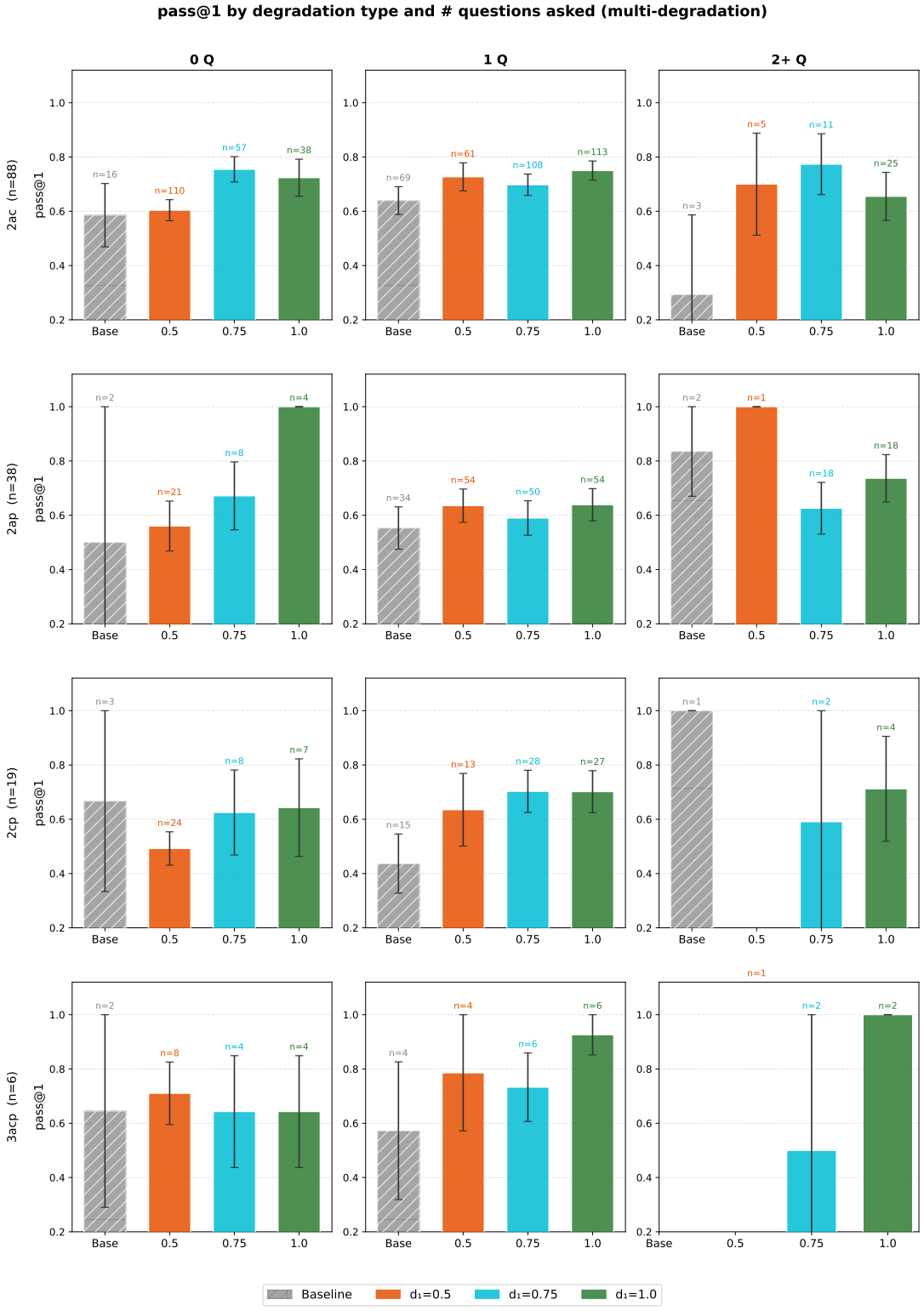

Figure 8: pass@1 by degradation type and number of questions asked, for multi-degradation types (2ac, 2ap, 2cp, 3acp). Same layout and pooling as Figure 7. Cells with very small n (notably 3acp and the 2+ Q column) should be interpreted cautiously.

D Representative Episode Comparisons

Each episode below shows the degraded specification presented to the agent, followed by the conversation. All episodes are from the 417-problem held-out evaluation set under greedy decoding (temperature $= 0$).

Episode 1. Interaction format; targeted vs. edge-case questioning (HumanEval/133/prompt1p, type: 1p). The spec body is truncated mid-sentence; the task description is entirely missing.

Degraded Spec

```
def candidate(lst):
    """You are given a list of numbers.
    You need to return the sum of squared
    """
```

Baseline (pass@1 $= 0.00$, 1 question):

Turn 1 [ASK]

```
What should be the behavior if the input
    list is empty?
Should it return 0 or raise an error?
```

Simulator

```
The behavior for an empty input list is not
    specified
in the provided documentation.
```

Turn 2 [ANSWER]

```
0
```

Trained policy d_1=1.0 (pass@1 $= 1.00$, 1 question):

Turn 1 [ASK]

```
What is the task? Specifically, what should
    be done
with the list of numbers?
```

Simulator

```
The task is to return the sum of the squared
     numbers
in the given list after rounding each
    element to the
upper integer (ceiling) first.
```

Turn 2 [ANSWER]

```
import math
def candidate(lst):
    return sum(math.ceil(x) ** 2 for x in
    lst)
```

The baseline does not yet know what the function is supposed to compute, yet asks about an edge case, a question that is only meaningful once the core task is understood. It receives an uninformative answer and submits a literal `0`. The trained policy identifies that the *entire task description* is missing and recovers it in one targeted question.

Episode 2. Pareto efficiency: fewer questions, higher accuracy (HumanEval/45/prompt1p, type: 1p). The spec description is truncated to a single ambiguous word: the docstring reads only "`Given length of a side and high, return.`", omitting the return type and leaving `high` undefined. The baseline spends its question budget confirming the obvious (`high` = height), but its answer still fails; the d_1=0.5 policy answers immediately and correctly.

Degraded Spec

```
def triangle_area(a, h):
    """Given length of a side and high,
    return.
    """
```

Baseline (pass@1 $= 0.00$, 1 question):

Turn 1 [ASK]

```
Is the high meant to be the height of the
    triangle?
```

Simulator

```
Yes, in this context, 'high' refers to the
    height of the triangle.
```

Turn 2 [ANSWER]

```
0.5 * a * h
```

Trained policy d_1=0.5 (pass@1 $= 1.00$, 0 questions):

Turn 1 [ANSWER]

```
def triangle_area(a, h):
    return 0.5 * a * h
```

The baseline wastes its question budget confirming that `high` means height, information that does not change the solution, and then outputs a bare expression instead of a function definition, scoring 0. The trained policy infers the formula directly from the signature and returns a complete, correct function without asking anything. This is one concrete instance of the Pareto dominance shown in Figure 1.

Episode 3. Multi-degradation problems require more questions (HumanEval/126/prompt2ap, type: 2ap). The spec has two simultaneous degradations: the sort direction is ambiguous *and* the duplicate-handling rule is truncated. A single question can resolve at most one.

Degraded Spec

```
def candidate(lst):
    '''
    Given a list of numbers, return whether
     or not they are
    sorted in a specific order. If list has
     certain duplicate
    of the same number,
    '''
```

Baseline (pass@1 = 0.85, 1 question):

Turn 1 [ASK]

```
What is the specific order that the list
     should be sorted in?
```

Simulator

```
The list should be sorted in ascending order.
```

Turn 2 [ANSWER]

```
def candidate(lst):
    return lst == sorted(lst)   # misses
     duplicate rule entirely
```

Trained policy d_1=1.0 (pass@1 = 1.00, 2 questions):

Turn 1 [ASK]

```
1. What is the specific order the list
     should be sorted in?
2. How should the function handle lists with
      duplicate numbers?
```

Simulator

```
1. The list should be sorted in ascending
     order.
2. The function should return False if the
     list has more than
   1 duplicate of the same number.
```

Turn 2 [ANSWER]

```
def candidate(lst):
    if lst != sorted(lst):
        return False
    from collections import Counter
    counts = Counter(lst)
    return all(c <= 2 for c in counts.values
    ())
```

The baseline resolves the ambiguous sort direction but never reaches the truncated duplicate rule, producing a partially correct solution. The trained policy recognises both signals of missing information: the unspecified sort direction *and* the visibly cut-off duplicate sentence ("`If list has certain duplicate of the same number,`"), and addresses both in a single two-part question, recovering the complete specification. This directly illustrates why Figure 6 shows higher mean question counts on harder multi-degradation problems.

E System Prompts

The policy receives a minimal zero-shot system prompt instructing it to respond with either `[ASK]` followed by a clarifying question or `[ANSWER]` followed by Python code, with one action permitted per turn. We deliberately avoided few-shot examples in the policy prompt: providing demonstrations would risk biasing the policy toward particular question styles or strategies, confounding the learned behaviour with the choice of examples. The GPT-4o-mini oracle simulator receives a separate system prompt that grants it access to the full ground-truth specification and instructs it to answer only what is asked, without volunteering test cases or the solution.

The placeholder `{original_prompt}` in the oracle simulator prompt is filled at runtime with the ground-truth un-degraded problem specification.

Policy system prompt

```
You are a coding assistant. Given a coding
     task below, you must either:
  - Ask a clarifying question by responding
     with [ASK] followed by
    your question.
  - Write your Python solution by responding
     with [ANSWER] followed
    by the code.
Important rules:
  - Respond with ONLY one action per turn ([
    ASK] or [ANSWER]).
  - When you have enough information, write
    the code.
  - Do not explain your reasoning -- just
    output the action directly.
```

Oracle simulator system prompt (count mode)

```
You are a helpful assistant who holds the
     complete specification
for a coding problem.

Full specification:
{original_prompt}

Note: The agent may refer to the function by
      a different name
(e.g., "candidate"). Treat any function name
      the agent uses as
referring to the function described above.

Rules:
1. Answer ONLY the specific question(s) the
     agent asks.
   Do not volunteer extra information.
2. Do not reveal test cases or the full
     solution.
3. Keep your answer brief and factual.
4. After your answer, on a new line write
     EXACTLY:
   QUESTION_COUNT: N
   where N is the number of distinct atomic
     questions you identified
   in the agent's message.

Counting rules for N:
- Each distinct piece of information being
     requested = 1.
- "What is X and what is Y?" = 2.
- "What is X?" = 1.
- Conjunctions like "and", "also", "
     additionally" between separate
  requests = separate questions.
- "Describe the full behavior of X" where X
     is one concept = 1.
```