FROM PLANNING FAILURES TO GROUNDED ACTIONS THROUGH SYNTHETIC TRAINING ON AMBIGUOUS EMBODIED INSTRUCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied AI studies agents that interpret human instructions and translate them into grounded action sequences for interaction with physical or simulated environments. A major challenge is that human instructions are often ambiguous, containing vague references, missing attributes, pronouns, and distracting conversational context. Small Language Models (SLMs) are attractive for efficient local planning, but their performance drops substantially under such ambiguity, while inference time prompting provides only limited improvement. To solve this issue, we introduce **FAST-GEN**, a failure driven synthetic training data generation framework that converts ambiguity induced planning failures into targeted supervision for textual SLM planners. Unlike prior approaches that primarily address ambiguity at inference time or train on clean demonstrations, FAST-GEN explicitly uses planner failures to generate corrective, ambiguity focused training data. FAST-GEN generates grounded ambiguous planning examples, detects and categorizes failed plans using grounding checks and a critic, and feeds the identified failure type back into the pipeline to generate corrected trajectories that are retained as supervised finetuning data. On REI-Bench, FAST-GEN improves 1B to 4B SLMs by roughly 35 to 62 absolute percentage points over inference time prompting baselines and also outperforms proprietary LLMs like GPT-5. Direct FAST-GEN training on AmbiK improves commonsense and safety success by 13.2 and 7.7 points over zero shot transfer. In out of distribution evaluation, FAST-GEN improves CLARA macro-F1 by 6.14 points and AmbiK safety success by 5.8 points, boosts DialFRED execution success by 21.5 to 29.0 points, and reduces hazardous task risk on SafeAgentBench by up to 7.7 points. Overall, FAST-GEN provides a scalable recipe for training ambiguity robust SLM planners in embodied domain.

1 INTRODUCTION

Embodied task planning enables an intelligent agent to interpret natural language instructions and convert them into grounded action sequences that can be executed in a physical or simulated environment. This capability is fundamental to language driven embodied systems because successful interaction depends not only on generating valid actions, but also on correctly understanding what a user intends. The importance of this problem is reflected in sustained research attention across major robotics, vision, and language venues (Huang et al., 2022a; Ahn et al., 2022; Shridhar et al., 2019). A central challenge is that human instructions are frequently underspecified and rely on implicit context, omitted attributes, pronouns, and information introduced earlier in a conversation(Cao et al., 2026). Such ambiguity can change the intended target, destination, or required action sequence even when the language appears superficially simple (Rana et al., 2026; Xu et al., 2026a). Robustly resolving these ambiguities is therefore an important prerequisite for reliable interaction between humans and embodied agents(Fang et al., 2026).

Existing work has approached language based embodied planning from several directions. Large language models have been used as planners or reasoning modules because of their strong instruction understanding capabilities (Achiam et al., 2023; Ahn et al., 2022), while smaller language models provide an attractive alternative for settings where computational efficiency, low latency, and local deployment are important(Wang et al., 2026; Izzo et al., 2026). A separate line of work improves

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

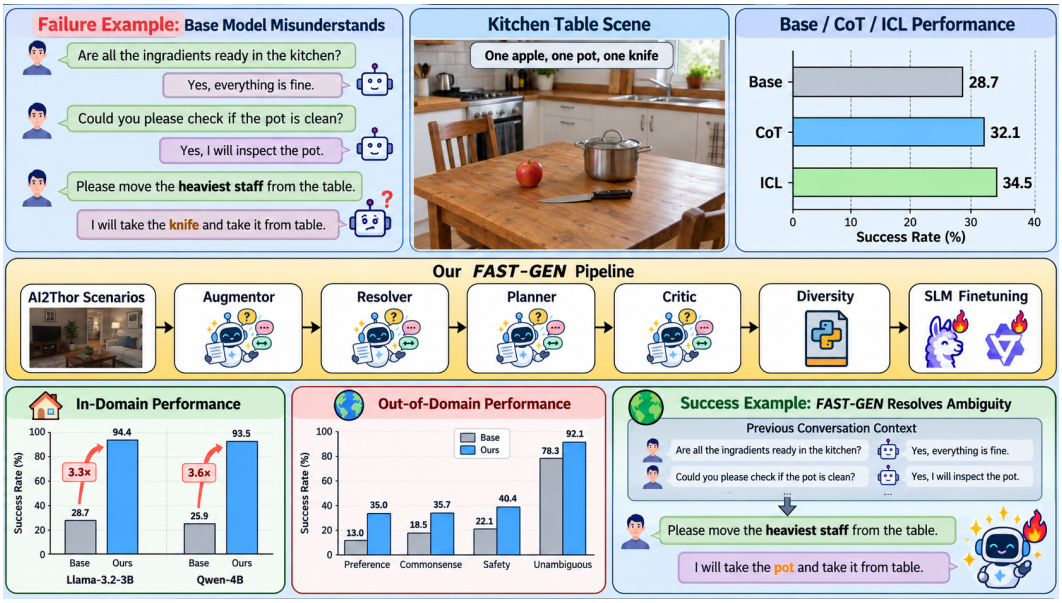


Figure 1: Ambiguous instructions can cause base SLMs to infer the wrong goal and generate invalid plans. Fine-tuning on FAST-GEN generated data enables SLMs to recover the intended goal and produce grounded action plans.

planning through inference time techniques such as Chain of Thought reasoning (Wei et al., 2022), in context demonstrations (Brown et al., 2020), and action aware prompting (Gao et al., 2024). These methods can improve reasoning without modifying model parameters, but they do not directly provide supervised signals that teach a planner how to recover from recurring ambiguity induced failures(Xu et al., 2026b). Existing synthetic data approaches for embodied planning also primarily focus on generating demonstrations from well specified tasks rather than constructing supervision around failures caused by vague or underspecified language(Liao et al., 2026). Consequently, there remains a need for training methods that explicitly identify how planners fail under ambiguity and transform those failures into targeted supervision.

We address this gap with FAST-GEN, a failure driven synthetic training data generation multiagent framework for ambiguity robust textual planning. FAST-GEN first analyzes baseline planning failures and organizes them into an eight category taxonomy covering action level, reference grounding, and conversational context errors. Using AI2-THOR (Kolve et al., 2017), the framework then constructs fresh grounded scenarios independently of the evaluation examples and introduces controlled linguistic ambiguity and contextual variation. A Resolver maps ambiguous instructions to grounded entities, while a Planner generates candidate action sequences that are checked against deterministic grounding constraints. The Critic uses the previously identified failure categories as a diagnostic vocabulary, identifies the failure type of incorrect candidate plans, and returns corrective feedback to the generation loop. In this way, FAST-GEN uses observed failure patterns to guide the creation of grounded, verified, and diversity balanced supervised fine tuning data without requiring manual annotation.

We evaluate FAST-GEN using Llama-3.2 (Grattafiori et al., 2024) and Qwen3 (Yang et al., 2025a) models ranging from 1B to 4B parameters across both in domain and out of domain ambiguity benchmarks. On REI-Bench (Jiang et al., 2025), FAST-GEN trained models achieve 92.6 to 98.1% success across nine ambiguity conditions and substantially outperform their base models and inference time prompting baselines. The resulting planners also outperform substantially larger directly prompted models, showing that increasing model scale alone does not resolve the ambiguity problem. Beyond REI-Bench, applying FAST-GEN directly to AmbiK improves ambiguity sensitive planning, while models trained only on REI-Bench style synthetic data transfer to AmbiK and CLARA without target-domain fine-tuning. Across these experiments, the main finding is that targeted supervision constructed from recurring planning failures is substantially more effective for ambiguity robustness than relying only on larger models or inference time reasoning strategies. Our contributions are:

- **Failure-Driven Synthetic Supervision.** We introduce FAST-GEN, a grounded multiagent data generation framework that turns empirically observed ambiguity induced planning failures into targeted supervision. Rather than generating generic successful demonstrations, FAST-GEN uses 8 category failure taxonomy to diagnose newly generated plans, provide corrective feedback, and retain only grounded, verified, and diversity balanced trajectories for fine-tuning.

- **Large Gains with Compact Planners.** We show that failure targeted supervision dramatically improves 1B to 4B Llama-3.2 and Qwen3 planners across all nine REI-Bench ambiguity conditions, reaching 92.6 to 98.1% success. These compact fine-tuned models consistently outperform their base counterparts, inference time reasoning strategies, graph based planners, and substantially larger directly prompted models, demonstrating that targeted training can be more effective than model scaling or additional inference time reasoning.

- **Cross Benchmark Generalization and Safer Planning.** We demonstrate that FAST-GEN is not specific to REI-Bench: the same framework can be instantiated directly on AmbiK, while REI-Bench trained models transfer without target domain finetuning to AmbiK, CLARA, DialFRED, and SafeAgentBench. The resulting gains span ambiguity classification, executable household planning, and hazardous task risk reduction, showing that failure driven supervision transfers across task formulations, action spaces, and evaluation protocols.

2 Related Works

Embodied AI Based on LLMs Recent work explores LLM based embodied planning through graph informed action generation, symbolic constraints, and multimodal evaluation (Li et al., 2026; Yang et al., 2025b; Byrd et al., 2025; Liu et al., 2025a). Beyond direct prompting, Inner Monologue incorporates environmental feedback into closed loop planning (Huang et al., 2022b), while KnowNo uses conformal prediction to identify uncertainty and request human assistance (Ren et al., 2023). These approaches improve planning through feedback or clarification at inference time. FAST-GEN offers a complementary training based approach: it uses verified synthetic demonstrations to improve small LMs before deployment, motivated by computational and latency constraints of larger LMs.

Training of Embodied Agentic Systems Supervised finetuning, reinforcement learning, and knowledge distillation have been explored to improve embodied planning and train smaller language models (Ghasemipour et al., 2025; Pham et al., 2025; Boyle et al., 2025; Liu et al., 2026). Failure aware data generation also provides supervision beyond successful demonstrations; for example, INFUSER injects synthetic failures into trajectories to train embodied agents to recover from execution errors (Yoon et al.). FAST-GEN focuses on generating grounded planning supervision for instructions with contextual ambiguity. An offline analysis of planner failures establishes eight error categories spanning procedural, reference-grounding, and conversational context errors. These definitions guide a Critic within a separate synthetic generation pipeline that verifies and corrects candidate plans. The resulting data jointly supports procedural compliance and reference resolution, with ambiguity specific benefits assessed against simpler synthetic SFT baselines.

Benchmarks for Ambiguous Embodied Instructions Embodied benchmarks examine several forms of instruction ambiguity, including command clarity in CLARA (Park et al., 2023), contextual referring expressions in REI-Bench (Jiang et al., 2025), task intent ambiguity in AmbiK (Ivanova et al., 2025), visual disambiguation in VAGUE (Nam et al., 2024), and open vocabulary 3D ambiguity in Ambi3D (Ding et al., 2026). Complementary dialogue based settings include DialFRED (Gao et al., 2022), which enables agents to ask clarification questions, and TEACh (Padmakumar et al., 2022), which studies household task completion through situated dialogue. These benchmarks span ambiguity that can be resolved from the available dialogue and scene context as well as underspecified instructions that may require additional interaction. FAST-GEN focuses on the former: its generated examples preserve a well defined intended referent and executable plan that can be recovered from the provided context and grounded scene. We evaluate this through indomain training on REI-Bench and AmbiK, and through zero shot transfer to AmbiK, CLARA, DialFRED, and SafeAgentBench.

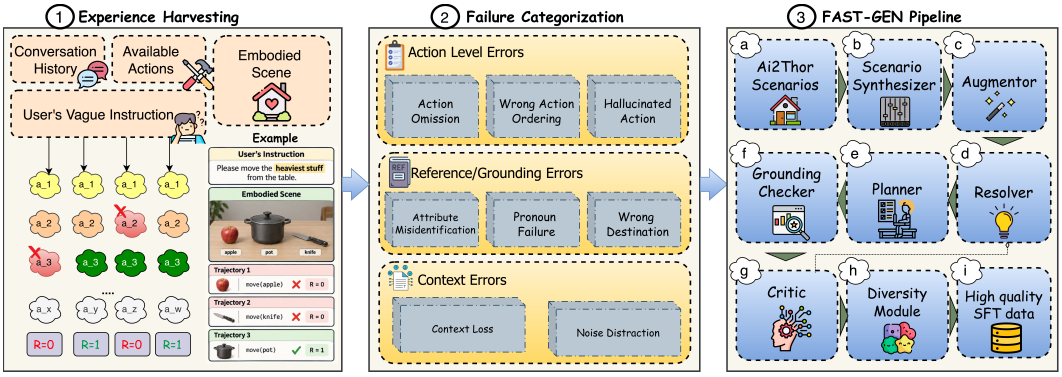


Figure 2: Overview of the failure driven data generation framework. **Stage 1:** baseline planners are evaluated on ambiguous embodied tasks to harvest success/failure trajectories. **Stage 2:** failures are grouped into eight categories spanning action, grounding, and context errors. **Stage 3:** new AI2-THOR scenes with controlled ambiguity are turned into candidate plans, grounding checked, diagnosed by a Critic using the Stage 2 categories, and diversity filtered into SFT data.

3 METHOD

3.1 STAGE 1: EXPERIENCE HARVESTING

We begin by characterizing how base small models of Llama, Qwen fail when instructions are linguistically ambiguous. Each evaluation instance contains four components: a conversation history D, an available action set $\mathcal{A}$, an embodied scene O, and a potentially vague user instruction I. Given $(D, I, O, \mathcal{A})$, a baseline planner generates an action trajectory $\hat{\pi} = (a_1, a_2, \ldots, a_T)$, where each $a_t \in \mathcal{A}$ denotes an executable action. The resulting trajectory is evaluated against the intended task outcome, producing a binary outcome $R \in \{0, 1\}$, where $R = 1$ denotes successful completion and $R = 0$ denotes a planning failure. For example, given the instruction "Please move the heaviest stuff from the table" and a scene containing an apple, a knife, and a pot, a planner may incorrectly select the apple or knife rather than the pot. By collecting such trajectories across ambiguous instructions and conversational settings, we obtain an empirical view of the recurring ways in which baseline planners misunderstand references, context, and action requirements.

3.2 STAGE 2: FAILURE CATEGORIZATION

We next analyze failed trajectories from Stage 1 and organize their errors into eight fine grained categories. The resulting taxonomy captures three broad sources of failure. **Action Level Errors:** These errors concern the actions contained in the generated plan and their local structure: *(i) Action Omission*, where a required action is missing; *(ii) Wrong Action Ordering*, where the required actions are present but appear in an invalid sequence; and *(iii) Hallucinated Action*, where the planner introduces an unsupported action. **Reference and Grounding Errors:** These failures arise when the planner associates an instruction with an incorrect entity or destination: *(iv) Attribute Misidentification*, where attributes such as size, color, or weight lead to selection of the wrong object; *(v) Pronoun Failure*, where expressions such as "it" or "them" are grounded to an incorrect referent; and *(vi) Wrong Destination*, where the correct object is manipulated but placed at the wrong destination. **Context Errors:** The remaining failures concern the use of conversational context: *(vii) Context Loss*, where information established earlier in the dialogue is ignored or forgotten; and *(viii) Noise Distraction*, where irrelevant conversational content distracts the planner from the actual task. This taxonomy provides the connection between failure analysis and synthetic data generation. Importantly, FAST-GEN does not receive the harvested trajectories or benchmark examples from Stage 1. Instead, only the definitions of errors are provided to the Critic as part of its system prompt. This allows the subsequent pipeline to recognize recurring failure modes while generating its training examples independently. Error categorization examples (A.1) and base model failure statistics (A.2) are in Appendix A.

3.3 STAGE 3: FAILURE DRIVEN SYNTHETIC DATA GENERATION

3.3.1 GROUNDED SCENARIO SYNTHESIS

We first construct a grounded task independently of any language model. Using AI2-THOR (Kolve et al., 2017), we define a scenario as $S = (\tau, o^*, r^*, O, \pi^*)$, where τ denotes the task type, o^* is

the intended target object, r^* is the target receptacle or destination, O represents the populated scene, and π^* is the corresponding gold action sequence. The Scenario Synthesizer samples valid objects, destinations, distractors, and task configurations from AI2-THOR primitives. The gold plan π^* is instantiated deterministically from the task specification rather than generated by an LLM. Consequently, the pipeline retains an explicit grounded reference against which subsequent resolutions and plans can be verified.

3.3.2 Controlled Ambiguity Generation

Given the grounded scenario S, the **Augmentor** converts the explicit task specification into a more realistic and potentially ambiguous human instruction. It generates an instruction I and a multiturn conversational context D while varying both referential explicitness and contextual noise. We consider explicit, mixed, and implicit referring expressions together with standard, noised, and shortened conversational contexts. Thus, an explicit instruction may directly name the target object, whereas an implicit instruction may refer to it through expressions such as "it", "that one", or an underspecified property such as "the heavier object". This controlled augmentation exposes the downstream planner to diverse forms of human like ambiguity while preserving the known grounded task.

3.3.3 Reference Resolution

The **Resolver** maps the ambiguous linguistic input back to the physical scene. Given (I, D, O), it predicts a target object $\hat{o}$ and destination $\hat{r}$ together with an explicit reasoning trace ρ: $R(I, D, O) = (\rho, \hat{o}, \hat{r})$. Because the scenario contains known targets o^* and r^*, resolution can be checked directly. A correctly grounded interpretation satisfies $\hat{o} = o^*, \hat{r} = r^*$. This step separates linguistic disambiguation from action generation and provides the planner with an explicit grounded interpretation of the user's intent.

3.3.4 Plan Generation and Grounding Verification

The **Planner** receives the the conversational context, vague instruction, Resolver output and scene description and generates an executable action sequence $P(D, I, \rho, \hat{o}, \hat{r}, O) = \hat{\pi}$. The Planner operates without access to the gold plan π^*, ensuring that the generated trajectory constitutes a genuine planning attempt rather than reproduction of a provided target sequence. The generated plan is then passed to a deterministic **Grounding Checker**. The checker verifies that every action belongs to the permitted skill vocabulary $\mathcal{A}$, that referenced objects and receptacles exist in O, and that the resulting sequence satisfies the required structural and physical constraints. Plans violating these conditions are rejected before entering the final corpus. This deterministic verification prevents physically invalid or hallucinated actions from being treated as successful demonstrations.

3.3.5 Failure Aware Critic and Corrective Feedback

The **Critic** provides the central failure driven component of FAST-GEN. It evaluates the instruction, conversation, resolved interpretation, generated plan, and grounded task specification and returns $C(I, D, \rho, \hat{o}, \hat{r}, \hat{\pi}, \pi^*) = (v, e, \phi)$, where $v \in \{\text{accept}, \text{reject}\}$ is the validity decision, $e \in \mathcal{E} \cup \{\text{none}\}$ identifies the failure category, and ϕ is natural language diagnostic feedback. The eight categories derived in Stage 2 appear *only as definitions in the Critic's system prompt*. No Stage 1 trajectory or benchmark instance is supplied to the Critic. The taxonomy therefore acts as a structured diagnostic vocabulary for identifying why a newly generated plan fails. When a plan is rejected, the Critic identifies the corresponding failure type and generates corrective feedback. This feedback is returned to the generation loop so that the ambiguous instruction can be reresolved and a corrected plan can be produced. For instance, if "heaviest" is grounded to an apple rather than a pot, the Critic can identify an *Attribute Misidentification* and guide the correction toward the intended object. In this way, FAST-GEN converts newly encountered planning errors into targeted corrective supervision rather than simply discarding all failed attempts.

3.3.6 Diversity Filtering and SFT Data Construction

Finally, the **Diversity Module** controls which accepted instances enter the supervised finetuning corpus. Rather than retaining all valid generations, we organize candidate examples according to $\mathcal{G} = \mathcal{T} \times \mathcal{R} \times \mathcal{X}$, where $\mathcal{T}$ denotes task types, $\mathcal{R} = \{\text{explicit}, \text{mixed}, \text{implicit}\}$ denotes referring expression regimes, and $\mathcal{X} = \{\text{standard}, \text{noised}, \text{short}\}$ denotes conversational context regimes. For each cell $g \in \mathcal{G}$, the module maintains a quota q and retains an example only when its corresponding cell is below this quota. We additionally compute $\kappa(x) = \text{SHA1}(I \| D)$ and reject previously observed hashes to reduce lexical duplication. Remaining deficits, $\{q - c(g)\}_{g \in \mathcal{G}}$, are fed back to the

synthesizer so that subsequent generation focuses on underrepresented combinations. The resulting corpus therefore contains grounded, verified, and diversity balanced examples covering different task types, ambiguity regimes, conversational conditions, and planning failures. These examples form the final high quality SFT dataset (Conversational context, instruction, resolver's CoT, planner's plan) used to train ambiguity robust planners. Our overall method is shown in Fig.2. Example of data generation steps (B.1), mathematics behind diversity (B.2), training data vs test data analysis (B.3), datagen model ablation (B.4),critic ablation (B.5),component ablation (B.6), effect of ambiguity targeted synthetic supervisionare (B.7), human audit of resolver and critic quality (B.8) and training data generation cost (B.9) in Appendix B.

4 Experimental Result

4.1 Experimental Setups

Models. We evaluate FAST-GEN on three open source instruction tuned LLM backbones: Llama-3.2-1B-Instruct (Liu et al., 2025b), Llama-3.2-3B-Instruct (Liu et al., 2025b), and Qwen3-4B-Instruct (Yang et al., 2025a). For each backbone, we compare the original instruction tuned model against its supervised finetuned variant trained on the FAST-GEN synthesized corpus. Hyperparameter ablation for these models is done in Appendix F. To assess whether ambiguity robustness can be obtained purely at inference time, we also evaluate several prompting baselines, including normal prompting, action aware prompting (AP) (Gao et al., 2024) , chain of thought prompting (CoT) (Wei et al., 2022), in context learning (ICL) (Brown et al., 2020), and task oriented contextual clarification (ToCC) (Jiang et al., 2025). This allows us to isolate the effect of failure driven SFT from prompt engineering improvements. Finetuning details are in Appendix C and cost token analysis of baselines are in Appendix D.

Benchmarks and Evaluation Metrics. We evaluate FAST-GEN under both in domain and out of domain settings. For in domain evaluation, we instantiate the generation pipeline separately for REI-Bench (Jiang et al., 2025) and AmbiK (Ivanova et al., 2025). REI-Bench covers nine ambiguity conditions formed by three reference types (explicit, mixed, implicit) and three context settings (standard, noised, short), while AmbiK evaluates unambiguous direct, preference, commonsense, and safety instructions. We report task success rate (SR) on both benchmarks. For out of domain evaluation, models trained exclusively on REI-Bench style FAST-GEN data are transferred without target domain finetuning to AmbiK, CLARA (Park et al., 2023), DialFRED (Gao et al., 2022), and SafeAgentBench (Yin et al., 2024). CLARA evaluates transfer from planning to ambiguity classification, for which we report macro-F1. On DialFRED, we evaluate all 424 validation tasks with human clarification dialogue and execute generated plans in AI2-THOR, reporting execution SR using ALFRED's goal checker, symbolic plan matching SR, goal condition completion (GC), and execution SR normalized by the goldplan ceiling. SafeAgentBench evaluates safety transfer on 300 detailed and 400 abstract hazardous instructions; we report hazardous task risk using both an LLM judge and AI2-THOR goal-state execution, together with rejection rate. These evaluations test whether failure driven supervision transfers across ambiguity types, task formulations, action spaces, and evaluation protocols.

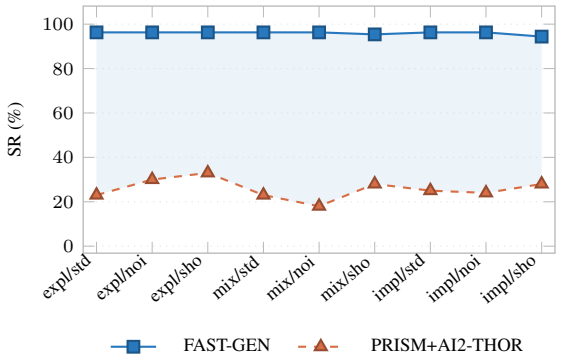


Figure 3: REI-Bench comparison on Llama-3.2-3B-Instruct. FAST-GEN consistently outperforms another data gen method PRISM+AI2-THOR across all nine ambiguity settings.

4.2 Main Results

In Distribution Result: Table 1 shows that FAST-GEN achieves the highest success across all nine REI-Bench conditions and three student backbones. Mean success increases from 21.6% to 93.9% for Llama-3.2-1B, 41.2% to 96.0% for Llama-3.2-3B, and 32.7% to 96.4% for Qwen3-4B. ICL is generally the strongest prompting baseline, reaching 61.0% on Qwen3-4B, while AP, CoT,

Table 1: REI-Bench success rates (%) across nine ambiguity conditions. We compare large model baselines and small models using inference time strategies and FAST-GEN finetuning. Within each small model block, **bold** denotes the best result and underlining denotes the second best.

Model	Variant	Explicit			Mixed			Implicit		
		Std	Noised	Short	Std	Noised	Short	Std	Noised	Short
Large-model direct-planning baselines										
Qwen3-235B-Inst.-2507	Base	38.9	38.0	52.8	44.4	39.8	51.9	46.3	35.2	48.1
Devstral2-123B	Base	55.6	55.6	55.6	53.7	50.0	47.2	54.6	53.7	47.2
GPT-OSS-120B	Base	50.9	59.3	49.1	49.1	47.2	50.9	46.3	52.8	46.3
Gemma4-31B-IT	Base	57.4	58.3	66.7	54.6	55.6	52.8	54.6	57.4	56.5
Muse-Glimmer-30B	Base	34.3	34.3	35.2	34.3	33.3	34.3	35.2	35.2	33.3
GPT-5	Base	81.5	81.5	80.6	76.9	80.5	75.5	77.3	80.5	81.5
Small-model inference and fine-tuning results										
Llama-3.2-1B-Inst.	Base	35.2	40.7	25.0	19.4	9.3	17.6	21.3	13.0	13.0
	+AP	39.8	38.9	28.7	14.8	8.3	20.4	16.7	11.1	16.7
	+CoT	29.6	29.6	22.2	11.1	5.6	12.0	13.0	7.4	13.9
	+ICL	39.8	51.9	33.3	25.0	25.9	29.6	19.4	34.3	22.2
	+ToCC	34.3	32.4	20.4	32.4	25.0	22.2	33.3	28.7	20.4
	FAST-GEN	**93.5**	**92.6**	**92.6**	**94.4**	**94.4**	**95.4**	**92.6**	**95.4**	**94.4**
Llama-3.2-3B-Inst.	Base	42.6	49.1	45.4	35.2	41.7	42.6	28.7	48.1	37.0
	+AP	40.7	49.1	44.4	35.2	46.3	45.4	28.7	48.1	42.6
	+CoT	39.8	41.7	36.1	35.2	34.3	25.0	37.0	35.2	38.0
	+ICL	41.7	51.9	50.9	39.8	46.3	44.4	28.7	50.9	45.4
	+ToCC	44.4	43.5	47.2	42.6	46.3	34.3	35.2	50.0	35.2
	FAST-GEN	**96.3**	**96.3**	**96.3**	**96.3**	**96.3**	**95.4**	**96.3**	**96.3**	**94.4**
Qwen3-4B-Inst.-2507	Base	36.1	39.8	26.9	31.5	35.2	25.9	35.2	35.2	28.7
	+AP	36.1	34.3	26.9	30.6	30.6	25.0	34.3	34.3	29.6
	+CoT	26.9	20.4	23.1	25.9	34.3	31.5	30.6	33.3	30.6
	+ICL	67.6	63.9	62.0	48.1	56.5	60.2	68.5	59.3	63.0
	+ToCC	36.1	38.0	31.5	25.9	30.6	26.9	33.3	32.4	29.6
	FAST-GEN	**98.1**	**98.1**	**98.1**	**97.2**	**93.5**	**94.4**	**97.2**	**96.3**	**94.4**

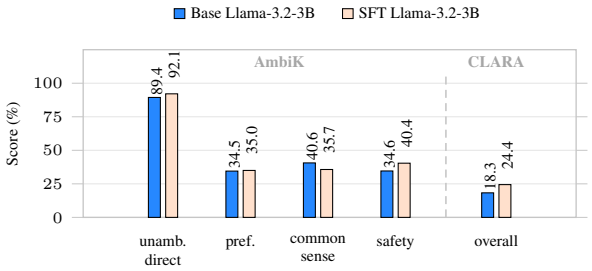


Figure 4: Out-of-domain results for Llama-3.2-3B-Inst. on AmbiK and CLARA.

and ToCC provide inconsistent gains. FAST-GEN also surpasses all evaluated large model baselines: GPT-5 (Singh et al., 2025), Gemma4-31B-IT (Team et al., 2026), Devstral2-123B (Rastogi et al., 2025), Qwen3-235B (Yang et al., 2025a) , Muse-Glimmer-30B (Meta Superintelligence Lab, 2026), and GPT-OSS-120B (Agarwal et al., 2025). Notably, the finetuned 3B to 4B students outperform their 235B data-generation teacher by 52.1 to 52.4 percentage points (More in Appendix G). To test whether the framework also works beyond the REI-Bench setting, we instantiate the complete FAST-GEN pipeline directly on AmbiK (results are in Appendix H). Under the same 1,800-example training budget, another data gen method PRISM (Ravichandran et al., 2025)+AI2-THOR achieves only 25.7% and 28.9% on Llama-3.2-3B and Qwen3-4B, respectively, compared with 96.0% and 96.4% for FAST-GEN (Fig. 3). We report the corresponding Qwen3-4B-Instruct results in Appendix I and visualization on simulator in REI-Bench is shown in Appendix N. We have also showed statistcal signifciance and scaling laws in Appendix L and M. We have also done comparison with Graph based planners as well as closed loop methods in Appendix Q and R.

Out of Distribution Result: On CLARA's 5,345-instruction macro-F1 evaluation, fine-tuning lifts Llama–3.2–3B-Inst. from 18.29% to 24.43% (Fig. 4, +6.14 absolute). Because macro-F1 weights every class equally, this gain reflects improved coverage of CLARA's minority categories rather than a shift toward the dominant label, suggesting that the disambiguation behavior learned on REI-Bench transfers to a broader taxonomy of clarification decisions. On AmbiK (Fig. 4), the same Llama–3.2–3B-Inst. SFT checkpoint improves over its instruction-tuned base on the categories where the benchmark expects a confident, well-grounded plan: *unambiguous direct* (89.4 to 92.1), *preferences* (34.5 to 35.0), and notably *safety* (34.6 to 40.4), with the safety gain being the largest single-category effect we observe on AmbiK. These cross-benchmark improvements—obtained without exposing the model to a single CLARA or AmbiK example during training—indicate that REI-Bench supervision encodes a form of disambiguation competence that generalizes beyond its own scene grammar, particularly for safety-critical and rare-class behaviors. Qwen Results on CLARA are mentioned in Appendix J. Fig. 5 compares FAST-GEN and PRISM+AI2-THOR on out-of-domain benchmarks for Llama-3.2-3B-Instruct. On AmbiK, FAST-GEN substantially outperforms PRISM+AI2-THOR across all ambiguity types, achieving success rates of 92.1%, 35.0%, 35.7%, and 40.4% compared to 35.1%, 8.7%, 12.4%, and 19.2% for PRISM+AI2-THOR on unambiguous-direct, preferences, common-sense, and safety settings, respectively. On CLARA, the two methods are competitive, with FAST-GEN achieving a macro-F1 of 24.4% compared to 24.0% for PRISM+AI2-THOR. These results show that FAST-GEN transfers better to out-of-domain ambiguity settings while maintaining comparable performance on the CLARA label space. We report the corresponding Qwen3-4B-Instruct results in Appendix I. OOD results in DialFRED and SafeAgentBench with visualization are in Appendix O and P.

4.3 Ablation Studies

Effect of per category training data To probe how the SFT gains depend on supervision volume, we re-train Llama-3.2-3B-Inst. while varying the number of training samples per REI-Bench category, $n \in \{50, 100, 150, 200\}$, keeping every other hyper-parameter unchanged (Fig. 6). Llama-3.2-3B-Inst. exhibits a sharp phase transition: 50, 100, and 150 samples per category produce comparable mean SRs of 81.80%, 75.71%, and 74.49%, respectively, with non-monotonic per-category behaviour. For example, `implicit_standard` bottoms out at 64.8% under n=150. Increasing the budget to n=200 raises the mean SR to 95.99% and flattens every category into the 94–96% band, gaining nearly 21.5 absolute points over n=150. We interpret this jump as evidence that REI-Bench's implicit and mixed ambiguity categories contain a long-tailed distribution of disambiguation patterns, requiring a critical mass of approximately 200 samples per category for the smaller Llama-3B-Inst. backbone to consolidate the desired behaviour. Practically, this motivates using n=200 samples per category as the default budget for Llama-class backbones. Qwen Ablation study is found in Appendix K.

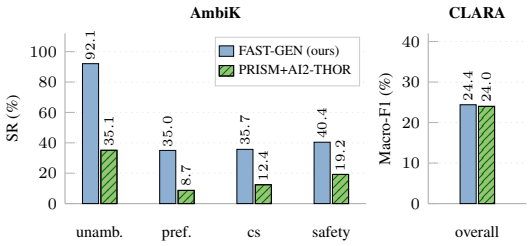


Figure 5: OOD comparison of FAST-GEN and PRISM+AI2-THOR on Llama-3.2-3B-Instruct. We report AmbiK success rate across ambiguity categories (unamb. = unambiguous direct, pref. = preferences, cs = common sense) and CLARA macro-F1 on the overall benchmark.

Number of ambiguity class in Training data We next isolate how many of REI-Bench's nine ambiguity classes ({explicit, mixed, implicit}×{standard, noised, short}) the model needs to observe during fine-tuning. Holding the per-class budget at 200 samples, we sweep the number of distinct classes in the training mixture: 1 class (200 samples), 4 classes (800 samples), and the full 9 classes (1,800 samples), while keeping all other hyper-parameters fixed (Fig. 7). Llama-3.2-3B-Inst. requires coverage of the full taxonomy: mean SR climbs 62.73 to 78.31 to 95.99 across the sweep. The 4 to 9 class jump (+17.68) is almost as large as the 1 to 4 class jump (+15.58), indicating that partial class coverage is insufficient for this backbone. The 4-class run still shows strong non-uniformity across cells, with explicit conditions at 79.6–84.3% and mixed/implicit conditions at 75.0–77.8%. Combined with the per-class budget ablation, this suggests that REI-Bench-quality SFT for Llama-

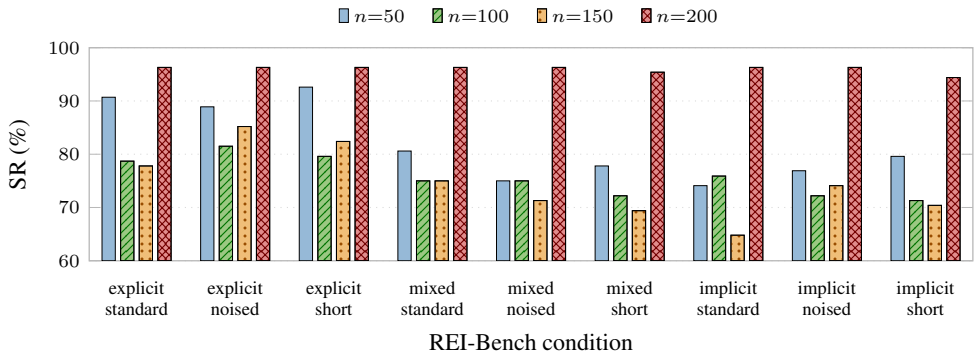


Figure 6: Llama-3.2-3B-Instruct SFT scaling on REI-Bench across all nine ambiguity conditions.

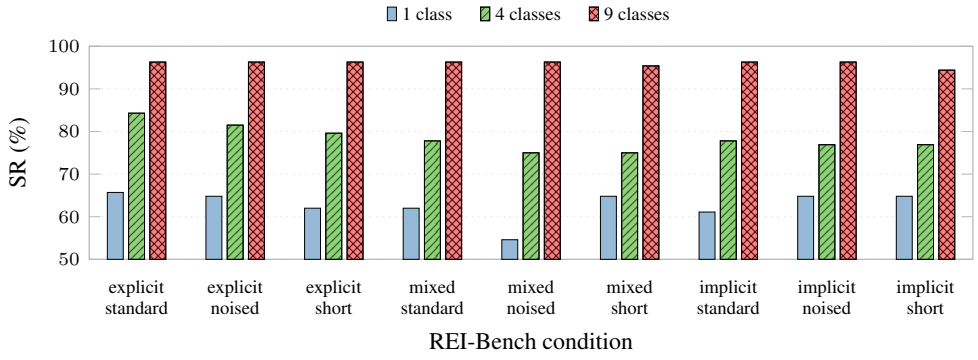


Figure 7: Llama-3.2-3B-Inst. class-coverage ablation on REI-Bench across all nine ambiguity conditions. Each selected class is trained with 200 examples; the average summary subplot is omitted.

3B-Inst. requires approximately 200 samples per class across all nine ambiguity classes. Qwen Ablation study is found in Appendix K.

5 Conclusion

We presented **FAST-GEN**, a scalable, annotation-free, failure driven framework for training small language model planners to handle ambiguous embodied instructions. FAST-GEN converts recurring planning failures into targeted supervision by combining grounded AI2-THOR scenario synthesis, controlled ambiguity generation, explicit reference resolution, deterministic grounding checks, failure aware critique, and diversity balanced data selection. Within the scope of textual planning with ambiguity that is resolvable from the provided dialogue and scene context, fine tuning 1B to 4B Llama-3.2 and Qwen3 models yields 92.6–98.1% success on REI-Bench and consistently outperforms base models, inference time prompting strategies, graph based planners, and substantially larger directly prompted models. The learned behavior also transfers beyond the training distribution, improving CLARA ambiguity classification, AmbiK safety sensitive planning, DialFRED executable task completion, and hazardous task risk on SafeAgentBench. Together, these results show that failure-targeted synthetic supervision can substantially improve ambiguity robustness and grounded action generation without increasing test-time model size or requiring additional inference-time reasoning. Future work will extend FAST-GEN to multimodal observations, active clarification for irreducibly underspecified instructions in richer simulated and physical environments.

AI use statement

This work uses large language models as components of the proposed FAST-GEN pipeline and as experimental baselines, as described in the Method and Experimental Setup sections. All generated training data are subjected to the grounding, critic, and diversity checks defined in our framework before being used for supervised fine-tuning. The authors take full responsibility for the experimental design, analysis, interpretation of results, and final manuscript.

Ethics statement

This work studies ambiguity-robust language-based planning in simulated embodied environments and does not involve human subjects, personally identifiable information, or physical robot deployment. The experiments are conducted on existing benchmarks and AI2-THOR-based synthetic scenarios. Although improved instruction understanding can make embodied agents more reliable, language-based planners may still produce incorrect or unsafe actions when deployed in real environments. Our results should therefore not be interpreted as establishing safety for autonomous physical deployment; such systems should include appropriate execution-time verification and human oversight in safety-critical settings.

Reproducibility statement

We provide detailed descriptions of the FAST-GEN pipeline, failure taxonomy, data-generation procedure, training setup, evaluation protocol, and hyperparameter choices. The appendix additionally includes the system prompts used by the LLM-based agents, training–test contamination analysis, component and data-generation ablations, statistical analysis across multiple seeds, and computational-cost statistics. We report the model variants, training-data sizes, learning rates, batch sizes, evaluation metrics, and benchmark splits required to reproduce the main experiments.

References

OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Moitinho de Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haim ing Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Made laine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Benjamin Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Sim'on Posada Fishman, Juston Forte, Is abella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Raphael Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Ryan Kiros, Matthew Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Li, Rachel Lim, Molly Lin, Stephanie L. Lin, Ma teusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel P. Mossing, Tong Mu, Mira Murati, Oleg Murk, David M'ely, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Ouyang Long, Cullen O'Keefe, Jakub W. Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alexandre Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Pondé de Oliveira Pinto, Michael Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack W. Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario D. Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie M. Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas A. Tezak, Madeleine

Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cer'on Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll L. Wainwright, Justin Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qim ing Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report. 2023. URL https://api.semanticscholar.org/CorpusID:257532815.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario M Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Jayant Joshi, Ryan C. Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego M Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, F. Xia, Ted Xiao, Peng Xu, Sichun Xu, and Mengyuan Yan. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:247939706.

Liam Boyle, Nicolas Baumann, Pavithhiren Sivasothilingam, Michele Magno, and Luca Benini. Robotxr1: Enabling embodied robotic intelligence on large language models through closed-loop reinforcement learning. *ArXiv*, abs/2505.03238, 2025. URL https://api.semanticscholar.org/CorpusID:278339497.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Thomas Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Ma teusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020. URL https://api.semanticscholar.org/CorpusID:218971783.

Grayson Byrd, Corban Rivera, Bethany Kemp, Meghan Booker, Aurora Schmidt, Celso de Melo, Lalithkumar Seenivasan, and Mathias Unberath. Constrained natural language action planning for resilient embodied systems. *ArXiv*, abs/2510.06357, 2025. URL https://api.semanticscholar.org/CorpusID:281892228.

Yuhao Cao, Xiao Liu, Yang Xie, Lu Liu, and Haoyao Chen. Sain: Structure-aware interactive navigation with active dialogue grounding for mobile robot. *arXiv preprint arXiv:2608.09196*, 2026.

Jiayu Ding, Haoran Tang, Hongbo Jin, Wei Gao, and Ge Li. 3d instruction ambiguity detection. 2026. URL https://api.semanticscholar.org/CorpusID:284597138.

Zeyu Fang, Yuxin Lin, Cheng Liu, Beomyeol Yu, Zeyuan Yang, Rongqian Chen, Taeyoung Lee, Mahdi Imani, and Tian Lan. Uncertainty mitigation and intent inference: A dual-mode human-machine joint planning system. *arXiv preprint arXiv:2603.07822*, 2026.

Jin Gao, Lei Gan, Yuankai Li, Yixin Ye, and Dequan Wang. Dissecting dissonance: Benchmarking large multimodal models against self-contradictory instructions. *ArXiv*, abs/2408.01091, 2024. URL https://api.semanticscholar.org/CorpusID:271693415.

Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S Sukhatme. Dialfred: Dialogue-enabled agents for embodied instruction following. *IEEE Robotics and Automation Letters*, 7(4):10049–10056, 2022.

Seyed Kamyar Seyed Ghasemipour, Ayzaan Wahid, Jonathan Tompson, Pannag R. Sanketi, and Igor Mordatch. Self-improving embodied foundation models. *ArXiv*, abs/2509.15155, 2025. URL https://api.semanticscholar.org/CorpusID:281393945.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Wenlong Huang, P. Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *ArXiv*, abs/2201.07207, 2022a. URL https://api.semanticscholar.org/CorpusID:246035276.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022b.

Anastasiia Ivanova, Eva Bakaeva, Zoya Volovikova, Alexey K. Kovalev, and Aleksandr I. Panov. Ambik: Dataset of ambiguous tasks in kitchen environment. *ArXiv*, abs/2506.04089, 2025. URL https://api.semanticscholar.org/CorpusID:279155321.

Riccardo Andrea Izzo, Gianluca Bardaro, and Matteo Matteucci. Btgenbot-2: Efficient behavior tree generation with small language models. *arXiv preprint arXiv:2602.01870*, 2026.

Chenxi Jiang, Chuhao Zhou, and Jianfei Yang. Rei-bench: Can embodied agents understand vague human instructions in task planning? *ArXiv*, abs/2505.10872, 2025. URL https://api.semanticscholar.org/CorpusID:278714946.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Kumar Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *ArXiv*, abs/1712.05474, 2017. URL https://api.semanticscholar.org/CorpusID:28328610.

Xiang Li, Ning Yan, and Masood S. Mortazavi. Embodied task planning via graph-informed action generation with large lanaguage model. *ArXiv*, abs/2601.21841, 2026. URL https://api.semanticscholar.org/CorpusID:285139940.

Zeqin Liao, Peifan Ren, Zixu Gao, Hongyu Gong, Lianyu Hu, Wenbing Tang, Yuhong Nan, Zibin Zheng, and Yang Liu. Vlamotor: Test-guided enhancement of vision-language-action models via agent-baseddata synthesis. *arXiv preprint arXiv:2606.00053*, 2026.

Jinrui Liu, Bingyan Nie, Boyu Li, Yaran Chen, Yuze Wang, Shunsen He, and Haoran Li. Robogpt-r1: Enhancing robot planning with reinforcement learning. *ArXiv*, abs/2510.14828, 2025a. URL https://api.semanticscholar.org/CorpusID:282138891.

Yaru Liu, Ao Wang, and Nan Ye. V-cage: Context-aware generation and verification for scalable long-horizon embodied tasks. *ArXiv*, abs/2601.15164, 2026. URL https://api.semanticscholar.org/CorpusID:284917499.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025b.

Meta Superintelligence Lab. Muse Glimmer Model Card. Hugging Face, August 2026. URL https://huggingface.co/meta-models/Muse-Glimmer-30B.

Heejeong Nam, Jinwoo Ahn, Keummin Ka, Jiwan Chung, and Youngjae Yu. Vague: Visual contexts clarify ambiguous expressions. *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1537–1547, 2024. URL https://api.semanticscholar.org/CorpusID:276928069.

Anatoly O Onishchenko, Alexey K Kovalev, and Aleksandr I Panov. Lookplangraph: Embodied instruction following method with vlm graph augmentation. *arXiv preprint arXiv:2512.21243*, 2025.

Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 2017–2025, 2022.

Jeongeun Park, Seungwon Lim, Joonhyung Lee, Sangbeom Park, Minsuk Chang, Youngjae Yu, and Sungjoon Choi. Clara: Classifying and disambiguating user commands for reliable interactive robotic agents. *IEEE Robotics and Automation Letters*, 9:1059–1066, 2023. URL https://api.semanticscholar.org/CorpusID:259202592.

Quang P. M. Pham, Khoi T. N. Nguyen, Nhi H. Doan, Cuong Anh Pham, Kentaro Inui, and Dezhen Song. Smallplan: Leverage small language models for sequential path planning with simulation-powered, llm-guided distillation. *ArXiv*, abs/2505.00831, 2025. URL https://api.semanticscholar.org/CorpusID:278310557.

Akshat Rana, Peeyush Agarwal, KPS Rana, and Amarjit Malhotra. Sg-cot: An ambiguity-aware robotic planning framework using scene graph representations. *arXiv preprint arXiv:2603.18271*, 2026.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. *arXiv preprint arXiv:2307.06135*, 2023.

Abhinav Rastogi, Adam Yang, Albert Q Jiang, Alexander H Liu, Alexandre Sablayrolles, Amélie Héliou, Amélie Martin, Anmol Agarwal, Andy Ehrenberg, Andy Lo, et al. Devstral: Fine-tuning language models for coding agent applications. *arXiv preprint arXiv:2509.25193*, 2025.

Zachary Ravichandran, Ignacio Hounie, Fernando Cladera, Alejandro Ribeiro, George J Pappas, and Vijay Kumar. Distilling on-device language models for robot planning with minimal human intervention. *arXiv preprint arXiv:2506.17486*, 2025.

Allen Z Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, et al. Robots that ask for help: Uncertainty alignment for large language model planners. *arXiv preprint arXiv:2307.01928*, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10737–10746, 2019. URL https://api.semanticscholar.org/CorpusID:208617407.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Wenhao Wang, Yanyan Li, Long Jiao, and Jiawei Yuan. Ro-slm: Onboard small language models for robot task planning and operation code generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 34436–34460, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022. URL https://api.semanticscholar.org/CorpusID:246411621.

Hengxuan Xu, Fengbo Lan, Zhixin Zhao, Shengjie Wang, Mengqiao Liu, Jieqian Sun, Yu Cheng, and Tao Zhang. Affordance-aware interactive decision-making and execution for ambiguous instructions. *arXiv preprint arXiv:2602.05273*, 2026a.

Peiran Xu, Jiaqi Zheng, and Yadong Mu. Roboagent: Chaining basic capabilities for embodied task planning. *arXiv preprint arXiv:2604.07774*, 2026b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *ArXiv*, abs/2502.09560, 2025b. URL https://api.semanticscholar.org/CorpusID:276317279.

Sheng Yin, Xianghe Pang, Yuanzhuo Ding, Menglan Chen, Yutong Bi, Yichen Xiong, Wenhao Huang, Zhen Xiang, Jing Shao, and Siheng Chen. Safeagentbench: A benchmark for safe task planning of embodied llm agents. *arXiv preprint arXiv:2412.13178*, 2024.

Byungjun Yoon, Dongyoung Kim, Jaehyun Nam, Wook Jung, and Jinwoo Shin. Infuser: Injecting synthetic failures for self-correcting embodied agents.

A More Details on Failure Analysis

A.1 Error Categorization Details

We have found 8 categories of errors from running REI-Bench on Llama, Qwen models. The details are as follows:

1 Action Omission

One or more **critical steps** required by the ground-truth plan are entirely absent from the agent's output. The agent understands the object and destination but treats intermediate manipulation steps (e.g., heating, cleaning) as implicit background assumptions rather than explicit actions to emit. The plan is structurally short, often completing directly from `pick_up` to `put_down`, skipping mandatory sub-routines.

Instruction: "*Heat and serve the potato.*"

Gold Plan: `find(Potato)` → `pick_up(Potato)` → `find(Microwave)` → `open(Microwave)` → `heat(Potato, Microwave)` → `close(Microwave)` → `find(DiningTable)` → `put_down(Potato, DiningTable)` → `done`

Agent Output: `find(Potato)` → `pick_up(Potato)` → `find(DiningTable)` → `put_down(Potato, DiningTable)` → `done`

Agent picks and places the potato but omits the entire `heat` *sub-routine.*

2 Action Ordering

All required actions are **present** in the plan but are executed in the **wrong logical sequence**. The agent fails to model causal dependencies between actions (e.g., an object must be found before it can be picked up).Often driven by surface mention-order in the instruction: the agent plans actions in the order entities are named, not the order they must be performed.

Instruction: "*Locate and pick up the apple.*"

Gold Plan: `find(Apple)` → `pick_up(Apple)` → `done`

Agent Output: `pick_up(Apple)` → `find(Apple)` → `done`

Agent attempts `pick_up` *before* `find`*, violating the prerequisite dependency.*

3 Hallucinated Action

The agent generates action verbs that are **not in the REI-Bench skill vocabulary**, or invokes objects that do not exist in the current scene. This arises when the model defaults to world-knowledge priors from pre-training rather than constraining generation to the closed, permitted action set. This error is more frequent at smaller model scales, where the skill vocabulary has not been sufficiently memorised during fine-tuning.

Instruction: "*Use available skills to prepare the food.*"

Valid skills include: `heat`, `pick_up`, `find`, `put_down`, `open`, `close`

Agent Output: `find(Apple)` → `pick_up(Apple)` → **`cook(Apple)`** → `done`

`cook` *is not a valid REI-Bench skill; the agent fabricates it from culinary world knowledge instead of using* `heat`.

4 Attributive Misidentification

The agent selects the **wrong object** due to incorrect interpretation of a descriptor such as **color, size, or state** (e.g., clean vs. dirty). The descriptor is parsed but not used as a hard filter; instead, the model resolves to the most scene-frequent or most recently encoded object sharing the same head noun. The resulting plan is structurally valid but operates on the wrong physical entity throughout.

Instruction: "*Pick up the red fruit.*"

Scene objects: RedApple, GreenApple, Bowl, CounterTop

Gold Plan: `find(RedApple)` → `pick_up(RedApple)` → `done`

Agent Output: `find(`**`GreenApple`**`)` → `pick_up(`**`GreenApple`**`)` → `done`

Agent picks the green apple despite the explicit colour attribute in the instruction.

5 Pronoun Failure

The agent fails to **resolve an anaphoric reference** ("it", "them", "there") to the correct entity established earlier in the instruction or dialogue context. Results in the entire plan being executed on or directed toward the wrong object or location. Common when the referent is introduced in a prior dialogue turn, far from the action-generating step in the token sequence.

Instruction: "*Pick up the apple. Heat it.*"

Gold Plan: `find(Apple)` → `pick_up(Apple)` → `find(Microwave)` → `open(Microwave)` → `heat(`**`Apple`**`, Microwave)` → `close(Microwave)` → `done`

Agent Output: `find(Apple)` → `pick_up(Apple)` → `find(Microwave)` → `open(Microwave)` → `heat(`**`Potato`**`, Microwave)` → `close(Microwave)` → `done`

Agent failed to associate "it" with the apple and resolved the pronoun to a distractor object in the scene.

6 Noise Distraction

Ambiguous object names or **distractor objects** whose names overlap with the target cause the agent to select the wrong entity. The agent matches on the lexical head noun (e.g., "book") but does not resolve partial-name distractors (e.g., `BookShelf`) or irrelevant co-present objects. Typically observable when the scene contains objects whose names are substrings or superstrings of the target name.

Instruction: "*Find the book.*"

Scene objects: Book, BookShelf, Newspaper, Desk

Gold Plan: `find(`**`Book`**`)` → `done`

Agent Output: `find(`**`BookShelf`**`)` → `done`

Agent targets `BookShelf` *because "book" appears as a prefix in its name, failing to distinguish the exact target from a containing object.*

7 Context Loss

The agent **fails to maintain task context** across multiple steps in a long-horizon plan, leading to hallucinated targets or arbitrary object substitution mid-execution. The agent begins the task correctly but forgets the original target partway through, switching to a random or scene-salient object. Particularly prevalent under implicit instructions where the target is established in a prior dialogue turn rather than the immediate instruction.

Context memory: "*User: I left the tomato on the counter. It needs to be chilled.*"

Instruction: "*Can you take care of it?*"

Gold Plan: `find(`**`Tomato`**`)` → `pick_up(`**`Tomato`**`)` → `find(Fridge)` → `open(Fridge)` → `cool(`**`Tomato`**`, Fridge)` → ···

Agent Output: `find(`**`Apple`**`)` → `pick_up(`**`Apple`**`)` → `find(Fridge)` → `cool(`**`Apple`**`, Fridge)` → ···

Agent ignores the dialogue-established target and substitutes `Apple` *— the first coolable object encountered in the scene list.*

8 WRONG DESTINATION

The target object is **correctly identified and manipulated** throughout the plan, but is placed at the **wrong final receptacle**. The agent resolves the destination using stereotypical spatial priors from pre-training (e.g., food goes on the counter) rather than the explicit or contextually specified location. Plans are otherwise complete and well-ordered; only the `put_down` argument is wrong.

Instruction: "*Put it in the fridge.*"

Gold Plan: ··· → `find(`**`Fridge`**`)` → `open(Fridge)` → `put_down(Apple, `**`Fridge`**`)` → `close(Fridge)` → `done`

Agent Output: ··· → `find(`**`CounterTop`**`)` → `put_down(Apple, `**`CounterTop`**`)` → `done`

Agent places the object on the counter despite the instruction explicitly specifying the fridge as the destination.

A.2 BASE MODEL FAILURE STATISTICS

LLAMA MODELS

We analyse the baseline REI-Bench results and identify **8 error categories** that account for all observed instruction-following failures across the 2 Llama models. They are: *(1) Action Omission, (2) Action Ordering, (3) Hallucinated Action, (4) Attributive Misidentification, (5) Pronoun Failure, (6) Noise Distraction, (7) Context Loss,* and *(8) Wrong Destination*. Details of them are in Appendix A.1. As shown in Figure 8, the error distribution differs noticeably between Llama 3.2 3B Inst and Llama 3.2 1B Inst. For Llama 3.2 3B Inst, errors are dominated by *Action Omission* at 47.20%, followed by *Action Ordering* at 37.41%. This suggests that the larger model most often fails by leaving out required actions or placing actions in the wrong sequence. Smaller categories include *Attributive Misidentification* at 6.12%, *Hallucinated Action* at 5.24%, *Pronoun Failure* at 2.62%, and *Wrong Destination* at 1.40%. In contrast, Llama 3.2 1B Inst shows a more distributed error pattern. Its largest category is *Action Ordering* at 36.61%, followed by *Hallucinated Action* at 22.44%, *Action Omission* at 20.21%, and *Attributive Misidentification* at 17.32%. This indicates that the smaller model not only struggles with ordering and omission, but also produces more hallucinated actions and attribution errors than the 3B model. Minor error categories include *Pronoun Failure* at 2.76%, *Wrong Destination* at 0.39%, *Noise Distraction* at 0.13%, and *Context Loss* at 0.13%.

QWEN MODELS

Figure 9 shows the error distribution across Qwen-family models. Action omission is the dominant failure mode across all variants, accounting for 76.44% of errors for Qwen3-4B-Instruct-2507 and remaining the largest category for Qwen 3 14B (56.16%), Qwen 3 8B (46.38%), and Qwen 3 32B (47.55%). As model size increases, the share of hallucinated actions becomes more prominent, reaching 37.75% for Qwen 3 32B, while pronoun-related failures also remain non-trivial across the stronger models.

B METHOD RELATED DETAILS

B.1 EXAMPLE OF DATA GENERATION STEPS

Stage 1: Programmatic Scenario Synthesis. For $\tau =$ `heat_and_place`, the synthesizer samples a physically heatable target and a valid placement receptacle, populates the scene with distractors, and instantiates π^* from the AI2-THOR template.

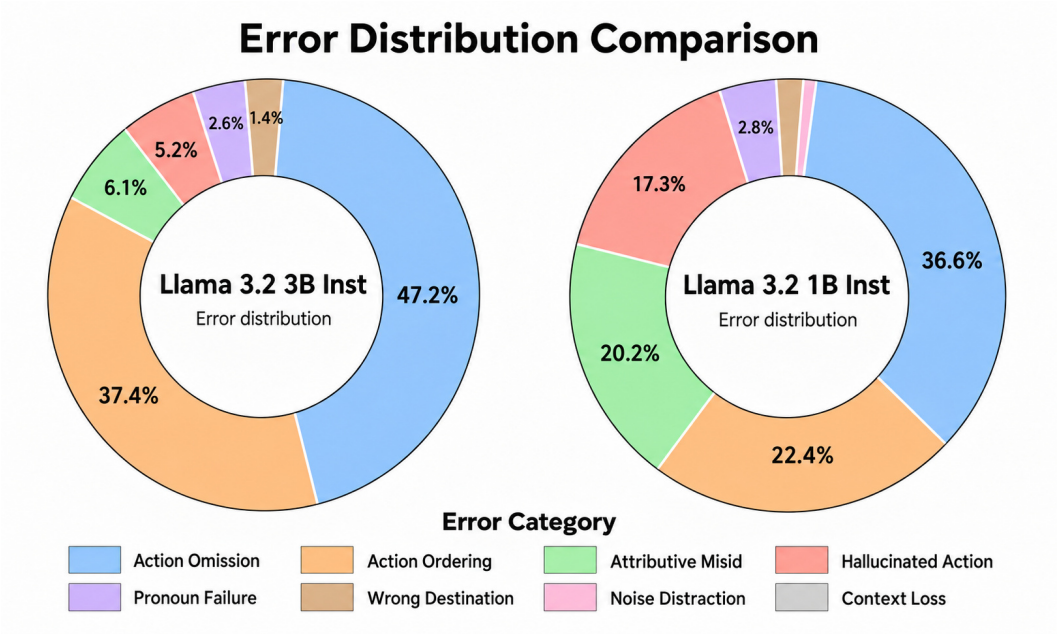


Figure 8: Error category distribution for Llama 3.2 3B Inst and Llama 3.2 1B Inst.

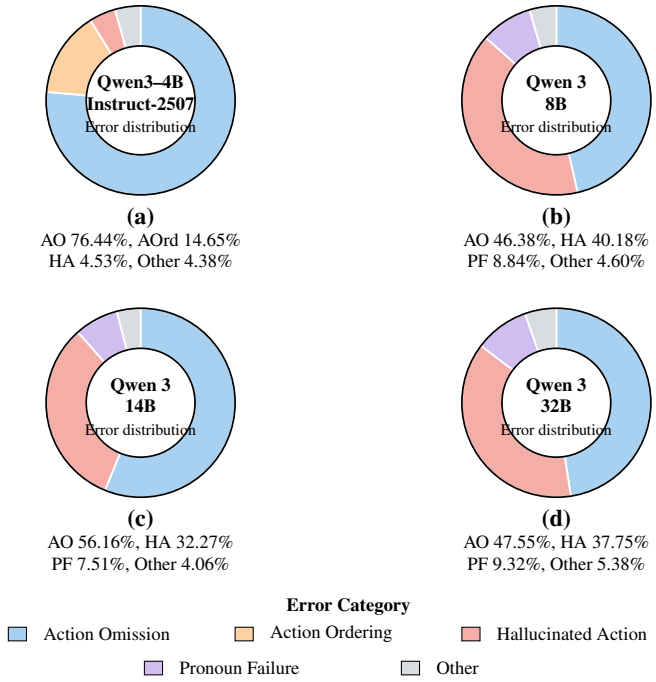


Figure 9: Critic-derived error distributions across Qwen-family models, ordered by model size. For readability, minor categories are grouped into *Other*. AO denotes action omission, AOrd denotes action ordering, HA denotes hallucinated action, and PF denotes pronoun failure.

τ `heat_and_place`
o^* `Bread`
r^* `CounterTop`
$\mathcal{O}$ `Bread, CounterTop, Microwave, Potato, ButterKnife, CreditCard, Dresser, Laptop`
π^* `find bread; pick_up bread; find microwave; open microwave; heat bread; close microwave; find counter top; put_down bread counter top; done`

The target is drawn only from the task-consistent pool (Bread $\in$ HEATABLES), the appliance (Microwave) is injected as a required scene object, and distractors (Potato, Laptop, ...) are sampled from the remaining object set. Since π^* is emitted symbolically from the template, it is correct by construction.

Stage 2: Augmentor Holding the physical spec from Stage 1 fixed (τ = heat_and_place, o^* = Bread, r^* = CounterTop), the Augmentor instantiates the *explicit/standard* cell of the difficulty matrix:

> **Dialogue context $\mathcal{D}$:**
> *Human: I just got back from the bakery with a fresh loaf of bread.*
> *Robot: Welcome back — I can see the bread on the table.*
> *Human: It went a little cold on the way home.*
> *Robot: Understood; you'd like the bread warmed up.*
>
> **Instruction $\mathcal{I}$:** "Heat the bread, then put it on the counter top."

Crucially, only $(\mathcal{I}, \mathcal{D})$ are generated; the target object, receptacle, and gold plan π^* remain untouched. Sliding the same spec to the *implicit/noised* cell instead yields "*Warm it up and leave it on the counter*," where the referent "it" must be resolved from the dialogue and a distractor object (Potato) is woven into the conversation — exercising the full spectrum of referential vagueness against a single, verifiable ground truth.

Stage 3: The Resolver Fed the harder *implicit/noised* cell from Stage 2 ($\mathcal{I}$ = "*Warm it up and leave it on the counter*", with Potato woven in as a distractor), the Resolver emits a structured resolution:

```
{
  "reasoning": "The instruction uses the pronoun 'it' with no explicit
noun. The dialogue establishes the referent — the human says 'I just got
back from the bakery with a fresh loaf of bread' — so 'it' resolves to
the bread, not the potato mentioned in passing. 'the counter' names the
destination.",
  "target": "Bread",
  "destination": "CounterTop"
}
```

The chain ρ exposes the disambiguation logic, while $(\hat{o}, \hat{r}) = (\texttt{Bread}, \texttt{CounterTop})$ grounds the downstream plan. Since $\hat{o} = o^*$ and $\hat{r} = r^*$, this resolution is correct; had the model latched onto the distractor (Potato), the mismatch would have automatically yielded a contrastive failure signal against the fixed ground truth.

Stage 4 & 5: Plan Generation and Grounding Verification. Given only the Resolver's output $(\hat{o}, \hat{r}) = (\texttt{Bread}, \texttt{CounterTop})$,the reason ρ, the instruction, the conversation and the scene $\mathcal{O}$—but *not* the gold plan π^*—the Planner regenerates the following plan $\hat{\pi}$:

```
{"plan": [
  "find bread",
  "pick_up bread",
  "find microwave",
  "open microwave",
  "heat bread",
  "close microwave",
  "find counter top",
  "put_down bread counter top",
  "done"
]}
```

The Grounding Checker $\mathcal{G}$ then verifies $\hat{\pi}$ against three hard constraints. First, every verb must lie in the AI2-THOR skill vocabulary:

$$\mathcal{A} = \{\texttt{find}, \texttt{pick_up}, \texttt{put_down}, \texttt{open}, \texttt{close}, \texttt{toggle_on}, \texttt{toggle_off}, \texttt{slice}, \texttt{clean}, \texttt{heat}, \texttt{cool}, \texttt{examine}, \texttt{done}\}.$$

Second, every object argument (bread, microwave, counter top) must resolve to an entity in $\mathcal{O}$. Third, the sequence must terminate in done. All checks pass, so the row is retained. Had the model

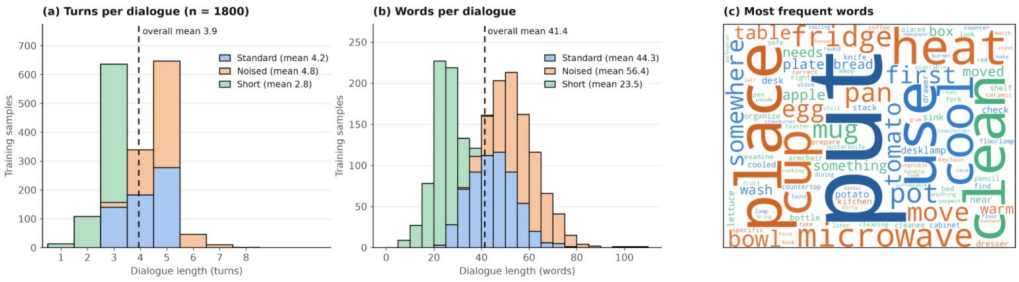


Figure 10: The 1,800-example training data set. **(a)** Dialogue length in turns and **(b)** in words, stacked by context type (standard, noised, short); the dashed line marks the overall mean. **(c)** Most frequent words in the instructions and dialogues, after removing stopwords and speaker labels; larger and darker words are more frequent.

emitted a hallucinated step such as `toast bread` (verb $\notin \mathcal{A}$) or `put_down bread oven` (object $\notin \mathcal{O}$), $\mathcal{G}$ would have discarded the plan outright, guaranteeing that the surviving dataset is physically executable.

Stage 6: The Critic The Critic ingests the full tuple $(\mathcal{I}, \mathcal{D}, \rho, \hat{o}, \hat{r}, \hat{\pi}, \pi^*)$ and renders a verdict. For the resolved plan from Stage 4 — which correctly heats the `Bread` and places it on the `CounterTop` — it returns a positive demonstration:

```
{"verdict": "accept",
  "error": "none",
 "diagnostic": "The pronoun 'it' is correctly grounded to the bread via
the bakery mention; the heating sub-sequence and final placement match
the instruction."}
```

Conversely, a counterfactual plan that latched onto the distractor — heating the `Potato` woven into the noised dialogue — is rejected with a fine-grained label drawn from the error taxonomy $\mathcal{E}$:

```
{"verdict": "reject",
 "error": "pronoun_failure",
 "diagnostic": "The pronoun 'it' was resolved to the potato distractor;
it should refer to the bread introduced earlier in the dialogue."}
```

The eight categories of $\mathcal{E}$ — *pronoun_failure*, *attributive_misid*, *noise_distraction*, *context_loss*, *action_ordering*, *action_omission*, *hallucinated_action*, and *wrong_destination* — let ϕ pinpoint *why* a plan fails, converting each rejected rollout into a targeted error-recovery signal rather than a mere binary negative.

Stage 7: The Diversity module Suppose the Critic has already passed enough *explicit/standard* `heat_and_place` instances to fill that cell's quota q. When a freshly accepted instance arrives carrying the same `Bread`/`CounterTop` spec and the instruction "*Heat the bread, then put it on the counter top*," $\mathcal{B}$ rejects it on two independent grounds: the cell count c has reached q, and the SHA-1 hash of this (instruction, dialogue) pair already lives in the seen-set $\mathcal{H}$. The deficit map, however, still shows the *implicit/noised* cell short by several rows, so the pipeline keeps generating for *that* cell instead — e.g. "*Warm it up and leave it there*," with a `Potato` distractor woven into the dialogue. In this way $\mathcal{B}$ converts the Critic's stream of accepted cases into a uniform allocation across the grid, preventing over-sampling of the trivial *explicit/short* corner while guaranteeing dense coverage of the hard, vague cells that matter most for robustness. Figure 10 shows the training set: dialogues average 3.9 turns (41.4 words), with noised contexts longest and short contexts shortest, and the most frequent words are task verbs and household objects (*put*, *place*, *heat*, *clean*, *microwave*).

B.2 MATHEMATICS BEHIND DIVERSITY

For each candidate row x with instruction $\mathcal{I}$ and dialogue $\mathcal{D}$, the diversity module computes a content key:

$$\kappa(x) := \text{SHA1}\Big(\underbrace{\text{lower}((\mathcal{I}))}_{\text{normalized instruction}} \| \texttt{0x7C} \| \underbrace{\text{lower}((\mathcal{D}))}_{\text{normalized dialogue}} \Big). \tag{1}$$

where $\|$ is byte concatenation and `0x7C` is the literal separator `|`. SHA-1 maps this arbitrary-length byte string to a fixed 160-bit digest, written as 40 hex characters. A row is admitted only if $\kappa(x) \notin \mathcal{H}$, after which $\mathcal{H} \leftarrow \mathcal{H} \cup \{\kappa(x)\}$.

Worked example from data generation. Take the explicit/standard heat-and-place row from the running Bread example:

- $\mathcal{I}$ = "Heat the bread, then put it on the counter top."
- $\mathcal{D}$ = "Human: I just got back from the bakery... Robot: Welcome back..."

The preimage string fed to SHA-1 is:

```
heat the bread, then put it on the counter top.|human: i just got back
from the bakery with a fresh loaf of bread. robot: welcome back, i can
see the bread on the table.
```

which hashes to:

$$\kappa(x_1) = \texttt{7d9f34b1...3475f21f}. \tag{2}$$

Case 1: near-duplicate is caught. A second row x_2 that the Augmentor emits with only capitalization and whitespace differences, e.g., "`HEAT the bread...`", normalizes to the same byte string, so

$$\begin{aligned} \kappa(x_2) &= \texttt{7d9f34b1...3475f21f} \\ &= \kappa(x_1) \Rightarrow \kappa(x_2) \in \mathcal{H} \Rightarrow \texttt{reject}. \end{aligned} \tag{3}$$

The collision is exact, not approximate: normalization via `strip().lower()` folds trivial surface variants onto one key, so they consume only one quota slot.

Case 2: genuine variation passes. The implicit/noised row x_3, e.g., "Warm it up and leave it there." with a Potato distractor, produces a different preimage and therefore a disjoint digest:

$$\begin{aligned} \kappa(x_3) &= \texttt{c9baa24d...c83523e4} \\ &\neq \kappa(x_1) \Rightarrow \texttt{accept}. \end{aligned} \tag{4}$$

Fig. 11 shows that the synthesized SFT training set spans a broad embedding space rather than collapsing into a narrow cluster. In the left panel, examples from the explicit, implicit, and mixed referring-expression types are interleaved across multiple PCA regions, indicating that the dataset covers diverse ambiguity forms rather than separating into isolated template-specific groups. In the right panel, the noised, short, and standard context variants also occupy overlapping but widely distributed regions, showing that linguistic surface form and context style introduce substantial variation while preserving shared task structure. Together, these PCA projections suggest that the 1,800 training instances provide diverse semantic and contextual coverage for fine-tuning.

B.3 TRAINING DATA AND REI-BENCH TEST-SET SEPARATION

Although the FAST-GEN training corpus and REI-Bench evaluation set share the same household domain, action space, and object inventory, no REI-Bench test instance is used during training. We therefore perform a complementary semantic and lexical audit to verify that the synthesized SFT corpus remains distinct from the held-out evaluation set. We compare the model-visible input, consisting of the dialogue context and final instruction, for all 1,800 synthetic training examples against all 972 REI-Bench test instances.

Fig. 13(a) provides a semantic view of this separation. We embed both corpora in the same representation space and project them using PCA. The synthesized training examples and REI-Bench

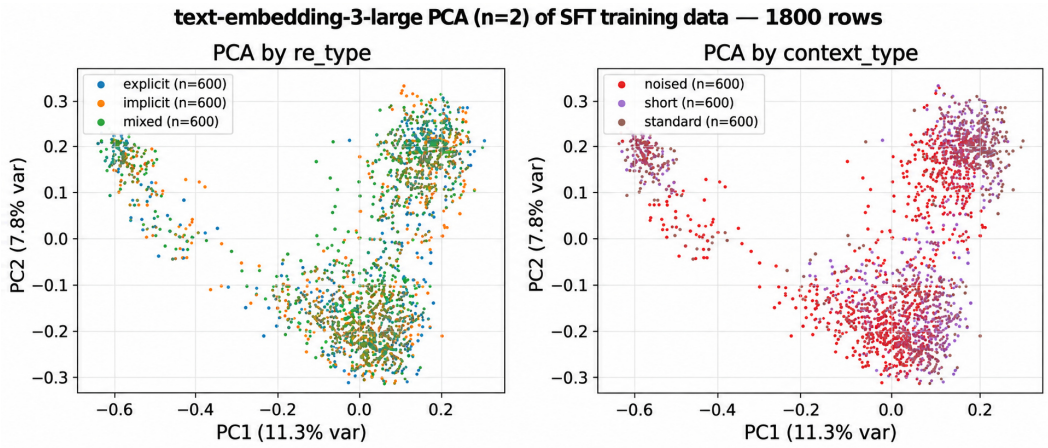


Figure 11: Training Data Diversity via PCA from text-embedding-3-large

test examples occupy visibly different regions, indicating that the generated corpus does not simply reproduce the evaluation distribution at the instance level. We further examine lexical overlap using n-gram matching (Fig. 13(b)). Short n-grams overlap frequently because both datasets necessarily share the same household vocabulary and restricted action primitives. However, this overlap rapidly decreases for longer spans. Only 13.5% of test instances share any 8-gram with the training set, while no test instance shares a 13-gram. Thus, the apparent overlap at short lengths primarily reflects the common task vocabulary rather than duplicated examples. Finally, we compute the maximum TF–IDF cosine similarity between every REI-Bench test input and the complete training corpus (Fig. 13(c)). We find $0/972$ exact duplicates, and the largest nearest-neighbour cosine similarity is only 0.526. Furthermore, no test instance reaches the 0.90 near-duplicate threshold. Together, the PCA, n-gram, and nearest-neighbour analyses provide complementary evidence that the FAST-GEN training corpus is distinct from the held-out REI-Bench evaluation set and that the reported gains are not explained by direct training-set contamination.

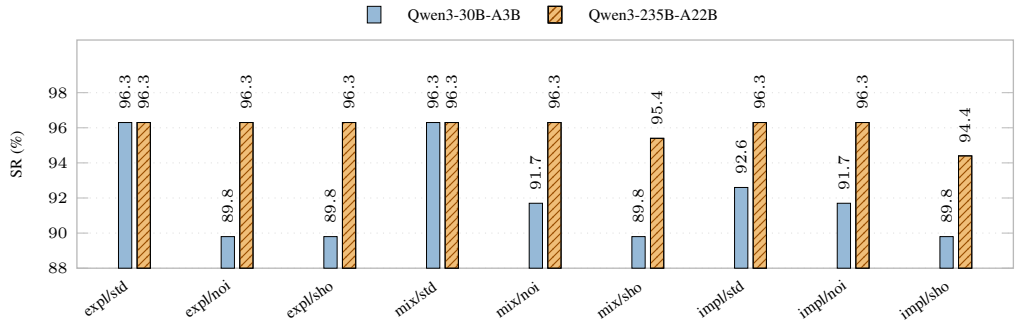
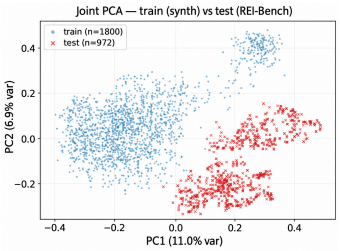


Figure 12: Ablation on data generation model size (30B vs. 235B) evaluated on REIBench.

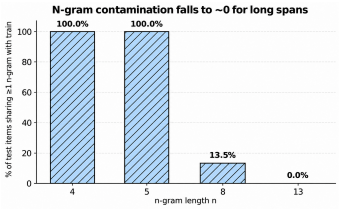
B.4 Data Generation Model Ablation

B.4.1 In Domain Result

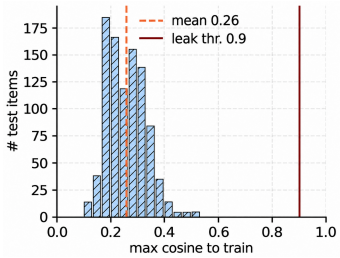
To investigate the impact of data generation model capacity, we generate 1.8K training samples using two variants of Qwen3: a 30B and a 235B parameter model. We then fine-tune Llama-3.2-3B-Instruct on each dataset via supervised fine-tuning (SFT) and evaluate on REIBench. As shown in Figure 12, the model trained on 235B-generated data consistently achieves higher success rates across nearly all conditions, particularly under implicit intent and short-prompt settings where the performance gap is most pronounced. This suggests that larger data generation models produce higher-quality training signals, and we therefore adopt the 235B model for all subsequent experiments.



(a) PCA projection of synthesized SFT training examples and REI-Bench test examples, showing separation in the shared embedding space.



(b) Percentage of test instances sharing at least one n-gram with training. Overlap decreases from 100% for short spans to 0% at $n = 13$.



(c) Maximum TF–IDF cosine similarity between each test instance and the training corpus. The maximum is 0.526, with $0/972$ examples exceeding the 0.90 near-duplicate threshold.

Figure 13: **FAST-GEN training and REI-Bench test-set separation. (a)** Semantic separation in the shared embedding space. **(b)** Long-span lexical overlap rapidly vanishes as n increases. **(c)** No exact or near-duplicate test inputs are found using TF–IDF nearest-neighbour similarity.

B.4.2 Out of Domain Result

To further validate the impact of data generation model size in an out-of-domain (OOD) setting, we evaluate on AmbiK and CLARA benchmarks using Llama-3.2-3B-Instruct as the SFT student. As shown in Figure 14, the model trained on data generated by the larger Qwen3-235B-A22B consistently outperforms its 30B counterpart on both benchmarks: on AmbiK, the success rate improves from 0.63 to 0.64, while on CLARA, the macro-F1 increases more substantially from 0.20 to 0.24. Although the gain on AmbiK is modest, the larger improvement on CLARA—an OOD benchmark with a different label space—suggests that higher-capacity data generation models produce richer and more transferable training signals. These results reinforce our choice of Qwen3-235B-A22B as the data generation model for all experiments.

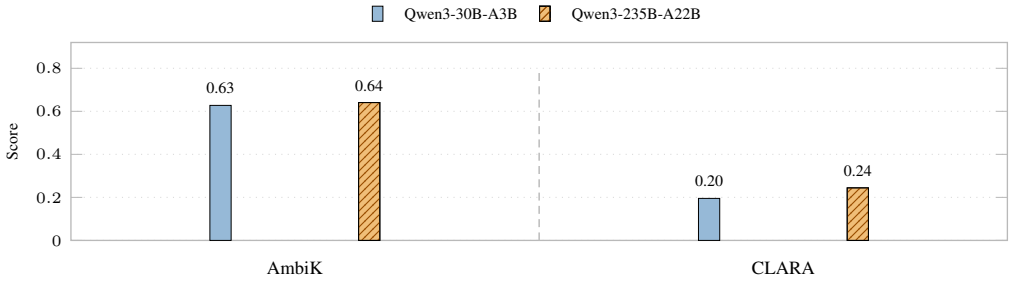


Figure 14: OOD ablation on data generation model size (30B vs. 235B) on AmbiK (SR overall) and CLARA (Macro-F1); Llama-3.2-3B-Instruct as SFT student.

B.5 Critic ablation with respect to error categories

To evaluate the contribution of structured error feedback in our critic model, we conduct an ablation study comparing a critic with 8 explicit error categories against a critic without any error categories, both trained on 1.8K SFT examples and evaluated on REIBench (972 items) using Qwen3-4B-Instruct. As shown in Figure 15, removing error categories consistently degrades performance across nearly all nine evaluation conditions, with the most pronounced drops observed in `mix/std` (96.3% to 88.9%) and `mix/noi` (97.2% to 91.7%), suggesting that mixed-intent instructions benefit most from fine-grained error signals. In contrast, conditions such as `expl/noi` and `expl/sho` show no degradation, indicating that explicit instructions are robust enough to be handled correctly even without categorical error feedback. Overall, the 8-category critic achieves 95.99% compared to 93.31% for the no-category variant (Δ=–2.67), confirming that structured error categories provide a meaningful and consistent improvement in training signal quality.

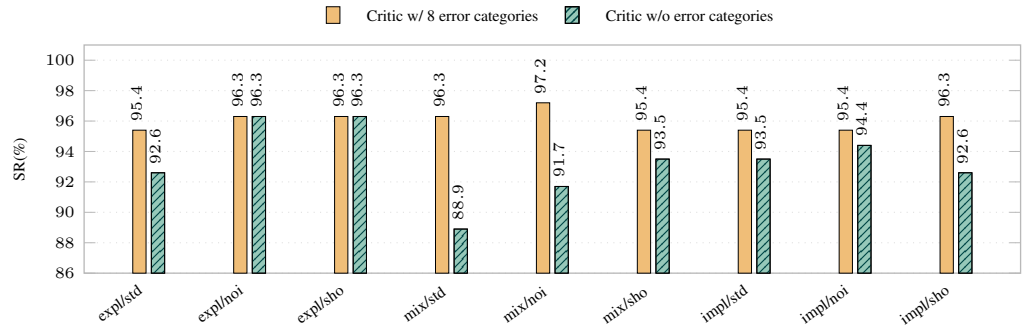


Figure 15: Critic error-category ablation on REIBench (Qwen3-4B-Instruct); removing error categories drops overall success rate from 95.99% to 93.31% (Δ=–2.67).

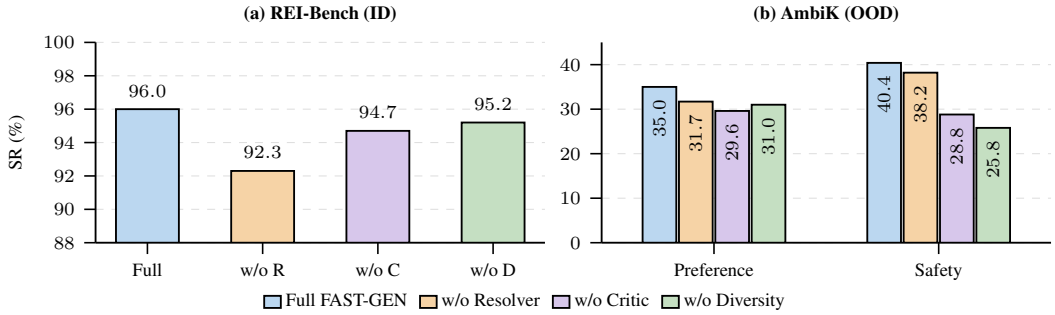


Figure 16: Component-removal ablations for Llama-3.2-3B-Instruct: (a) in-domain REI-Bench and (b) out-of-domain AmbiK (Preference and Safety).

B.6 Component Removal Ablation

To isolate the contribution of individual FAST-GEN components, we conduct component-removal ablations with a fixed training budget of 1,800 examples, using the same Llama-3.2-3B-Instruct backbone, hyperparameters, and training schedule. Figure 16 compares the complete pipeline with variants that remove the Resolver, Critic, or diversity targeting, evaluated in-domain on REI-Bench and out-of-domain on AmbiK without target-domain fine-tuning. On REI-Bench, the complete pipeline achieves 96.0% success. Removing the Resolver produces the largest decrease, to 92.3% (-3.7 percentage points), followed by removing the Critic (94.7%; -1.3 points) and diversity targeting (95.2%; -0.8 points). On AmbiK, the complete pipeline achieves 35.0% and 40.4% success on Preference and Safety tasks, respectively. Removing the Resolver reduces these scores to 31.7% and 38.2%, while removing the Critic yields 29.6% and 28.8%. Removing diversity targeting yields 31.0% and 25.8%. The Safety drops are particularly pronounced for the Critic and diversity ablations, at 11.6 and 14.6 percentage points, respectively. These results suggest that explicit reference resolution contributes most to in-domain performance, while failure-aware correction and diversity targeting are particularly important for generalization to out-of-domain Safety tasks.

B.7 Effect of Ambiguity-Targeted Synthetic Supervision

We next test whether FAST-GEN's gains can be explained simply by exposing the model to a large number of synthetic planning demonstrations. We compare the full pipeline against two simplified training-data controls under the same 1,800-example budget: training only on clean, unambiguous examples (unambiguous-only), and deterministic template-based supervision without linguistic variation or failure-driven correction (template-only). Fig. 17 reports results on REI-Bench and on the out-of-domain AmbiK benchmark.

On REI-Bench, unambiguous-only training reaches 92.5% success and template-only training reaches 91.5%, compared with 96.0% for full FAST-GEN. Removing ambiguity-focused supervision therefore reduces performance by 3.5 points, while removing both linguistic variation and failure-driven correction reduces it by 4.5 points. The condition-level results (Fig. 18) show where these losses

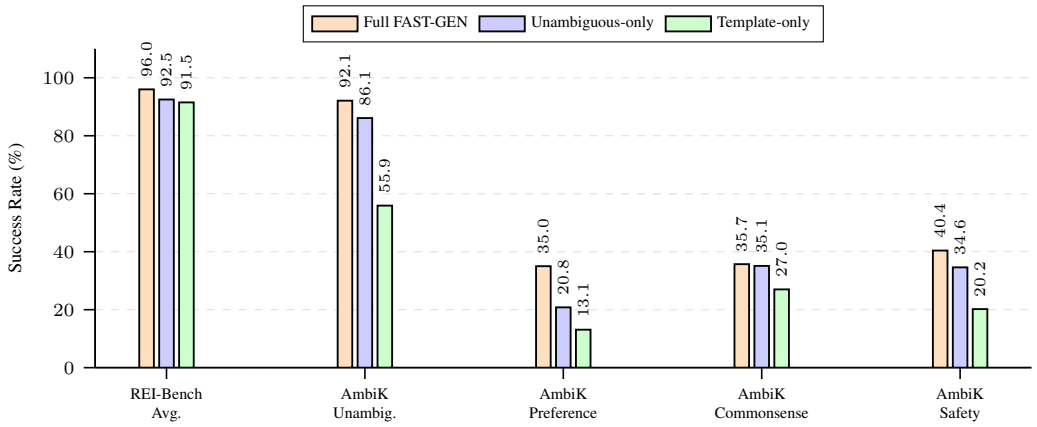


Figure 17: Synthetic-data controls on REI-Bench and out-of-domain AmbiK. Full FAST-GEN consistently outperforms unambiguous-only and template-only training, with substantially stronger transfer to preference, commonsense, and safety ambiguities on AmbiK.

arise. Template-only training performs well when the intended target is explicitly specified, but its performance deteriorates substantially for implicit instructions: success falls to 71.3%, 83.3%, and 87.0% on implicit-standard, implicit-noised, and implicit-short, respectively, compared with 96.3%, 96.3%, and 94.4% for full FAST-GEN. The largest gap, 25.0 points on implicit-standard, indicates that learning the basic action format alone is insufficient for resolving underspecified references.

The out-of-domain results on AmbiK show that the modest aggregate gaps on REI-Bench understate the differences between the training sets. On unambiguous AmbiK tasks, full FAST-GEN reaches 92.1%, compared with 86.1% for unambiguous-only and only 55.9% for template-only training. Because this 36.2-point drop occurs even when no ambiguity is present, it suggests that template-only supervision largely teaches the surface form of the training distribution rather than a transferable planning skill. On the ambiguous AmbiK categories, full FAST-GEN achieves 35.0%, 35.7%, and 40.4% on preference, commonsense, and safety ambiguities, compared with 20.8%, 35.1%, and 34.6% for unambiguous-only and 13.1%, 27.0%, and 20.2% for template-only. Relative to unambiguous-only training, ambiguity-focused supervision helps most for preference ($+14.2$ points) and safety ($+5.8$) ambiguities, while the two variants are nearly tied on commonsense ambiguities ($+0.6$), possibly because resolving these relies more on knowledge already present in the base model. Although absolute success on ambiguous AmbiK tasks remains modest for all variants, full FAST-GEN is best in every category, and its margin over template-only grows from 4.5 points in-domain to between 8.7 and 36.2 points out of domain. Together, these controls indicate that FAST-GEN's gains stem from the content of its supervision, namely ambiguity coverage, linguistic variation, and failure-driven correction, rather than from the volume of synthetic demonstrations alone.

B.8 Human Audit of Resolver and Critic Quality

Because FAST-GEN relies on LLM-based components during data generation, we additionally audit the quality of the Resolver and Critic outputs. We randomly sample approximately 150 generated examples and independently inspect them with three human annotators. For each example, the annotators assess (i) whether the Resolver identifies the intended target entity correctly and (ii) whether the Critic's accept/reject verdict is correct. The Resolver achieves approximately 99% agreement with the human judgments on target resolution, while Critic verdict correctness reaches approximately 96%. These high agreement rates indicate that the two LLM-based verification components introduce relatively little label noise into the retained SFT corpus and support the use of automated verification at scale.

B.9 Training Data-Generation cost

We additionally quantify the cost of generating the synthetic supervision used by FAST-GEN. Since Qwen3-235B is an open-source model that we run locally, our pipeline does not incur an external API fee; we therefore report generation cost in terms of teacher calls and token usage. Table 2 summarizes the statistics for the main REI-Bench training corpus and the in-domain AmbiK experiment.

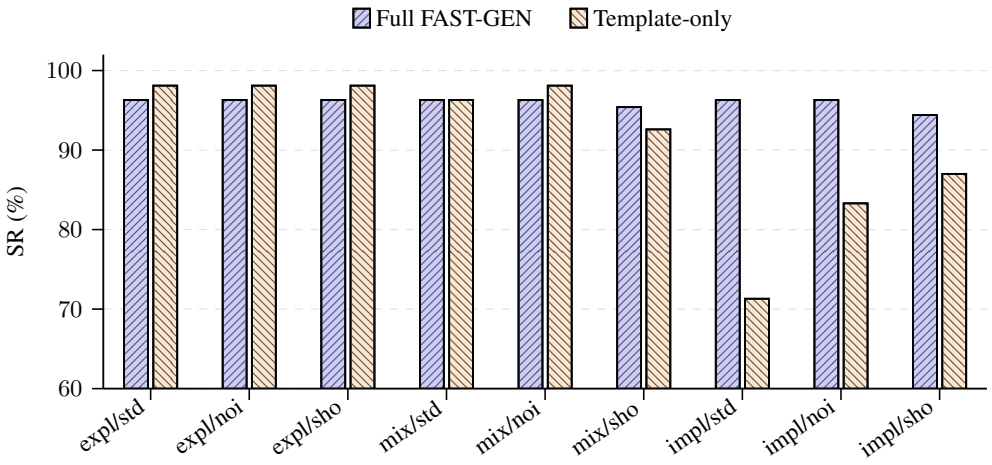


Figure 18: Condition-level comparison between full FAST-GEN and deterministic template-only supervision. Template-only data remains competitive on explicit conditions but degrades sharply under implicit ambiguity, showing the importance of ambiguity-targeted supervision.

Table 2: Data-generation cost of FAST-GEN for the main REI-Bench corpus and the in-domain AmbiK experiment. Token statistics correspond to calls to the Qwen3-235B teacher.

Metric	**REI-Bench**	**AmbiK**
Accepted examples	1,800	573
Generation attempts	3,600	1,296
Total prompt tokens	∼6.65M	∼2.19M
Total completion tokens	∼0.95M	∼0.47M
Total teacher tokens	∼7.60M	∼2.66M
Teacher tokens / accepted example	∼4.23K	∼4.64K

C Fine-tuning Details

Setup. We fine-tune every student model with LoRA. The base weights are frozen, and low-rank adapters are attached to all attention projections (`q`, `k`, `v`, `o`) and all MLP projections (`gate`, `up`, `down`). After training, the adapters are merged into the base weights, so inference costs the same as for the base model. The training set has 1,800 examples, each a dialogue history, a pending instruction, a target plan and Resolver's CoT. The loss is computed only on the assistant turn (the plan with the CoT); prompt and dialogue tokens are masked. Every model is trained with the same hyperparameters (Table 3); we do no per-model tuning.

Cost. Fine-tuning is cheap. Table 4 gives the wall-clock time and final training loss for each student model. The largest model (Qwen3-8B) trains in under 30 minutes on one GPU, and the smallest in under 5 minutes.

D Cost and Token Analysis of baselines

Figure 19 reveals a clear cost-quality tradeoff across the four prompting baselines that further motivates the SFT approach. ICL incurs the highest prompt token consumption for both models (∼839–840k tokens), owing to the inclusion of in-context exemplars, yet achieves the best inference throughput at 2.3 examples/s for Llama-3.2-1B-Inst. — a consequence of its relatively short completions (35k tokens). CoT and ToCC present the opposite profile: both generate substantially longer completions (282k and 151k tokens for 1B; 110k and 39k for 3B respectively) and suffer the worst evaluation times, with CoT requiring up to 3,840s and ToCC up to 2,646s across the two models, reducing throughput to as low as 0.2–0.4 examples/s. AP is the most lightweight baseline in terms of completion length and wall-clock time, yet as shown in Figure 19, it provides the least reliable performance gain. Crucially, none of these baselines escape the fundamental cost-scaling problem

Table 3: Fine-tuning hyperparameters

Hyperparameter	Value
Fine-tuning method	LoRA (adapters merged after training)
LoRA rank r / α	16 / 32
LoRA dropout	0.05
LoRA target modules	`q,k,v,o,gate,up,down_proj`
Optimizer	AdamW (β_1=0.9, β_2=0.999, ϵ=10^{-8})
Learning rate	Check Appendix F
LR schedule	Cosine decay, 3% linear warm-up
Weight decay	0.0
Effective batch size	Check Appendix F
Epochs	3 (336 optimizer steps)
Max sequence length	1,024 tokens
Loss	Cross-entropy on assistant (plan) tokens only
Precision	bf16, gradient checkpointing
Training examples	1,800
Random seed	42
Hardware	NVIDIA A100 80GB

Table 4: Wall-clock training time (1×A100 80GB, 3 epochs) and final training loss for each student model, using the hyperparameters in Table 3.

Base model	Time (min)	Final loss
Llama-3.2-1B-Instruct	4.5	0.47
Llama-3.2-3B-Instruct	8.4	0.43
Qwen3-0.6B	7.4	0.50
Qwen3-1.7B	11.8	0.51
Qwen3-4B	19.7	0.46
Qwen3-8B	29.8	0.43

of agentic deployment: even the cheapest configuration (ICL on 3B at 1.5 examples/s) accumulates prohibitive token budgets over hundreds of planning steps per episode. By contrast, once trained, our SFT models require no additional prompt augmentation, operate at native inference speed with minimal completion tokens, and eliminate per-query API overhead entirely — making FAST-GEN not only more accurate but substantially more economical for real-world embodied deployment.

E Reproduction of Comparison Baselines

None of the compared methods has a public REI-Bench implementation, so we re-implement each as a text-only planner on a shared harness, using the same two backbones (Llama-3.2-3B-Instruct and Qwen3-4B-Instruct-2507), the same 972-item evaluation set and the same scorer as our models, and never showing the gold plan. **KnowNo** (Ren et al., 2023) has the LLM propose four candidate next actions plus a "none of these" option, calibrates a conformal threshold ($\epsilon = 0.15$) on a held-out 30% split, and asks the human to name the referent whenever the prediction set is not a singleton, so it is scored on the remaining 684 items and reported without help. **Inner Monologue** (Huang et al., 2022b) emits one action at a time and feeds back a scene description, a symbolic success check and human object-level feedback on failures, with a no-feedback ablation of the same loop. **SayPlan** (Rana et al., 2023) searches a two-level scene graph built from the object list, writes the whole plan, and re-plans up to five times on feedback from a symbolic verifier. **LookPlanGraph** (Onishchenko et al., 2025) plans one reasoned action at a time over a memory graph, validates each action with a simulator, and stops after three consecutive infeasible actions or twenty actions, with the vision-based object discovery replaced by a textual oracle. **PRISM** data are generated with its upstream pipeline and LLM-Planner prompt, with the generating LLM swapped from GPT-4o to Qwen3-235B, in an AI2-THOR-scene variant (1,800 examples), then trained with our SFT recipe.

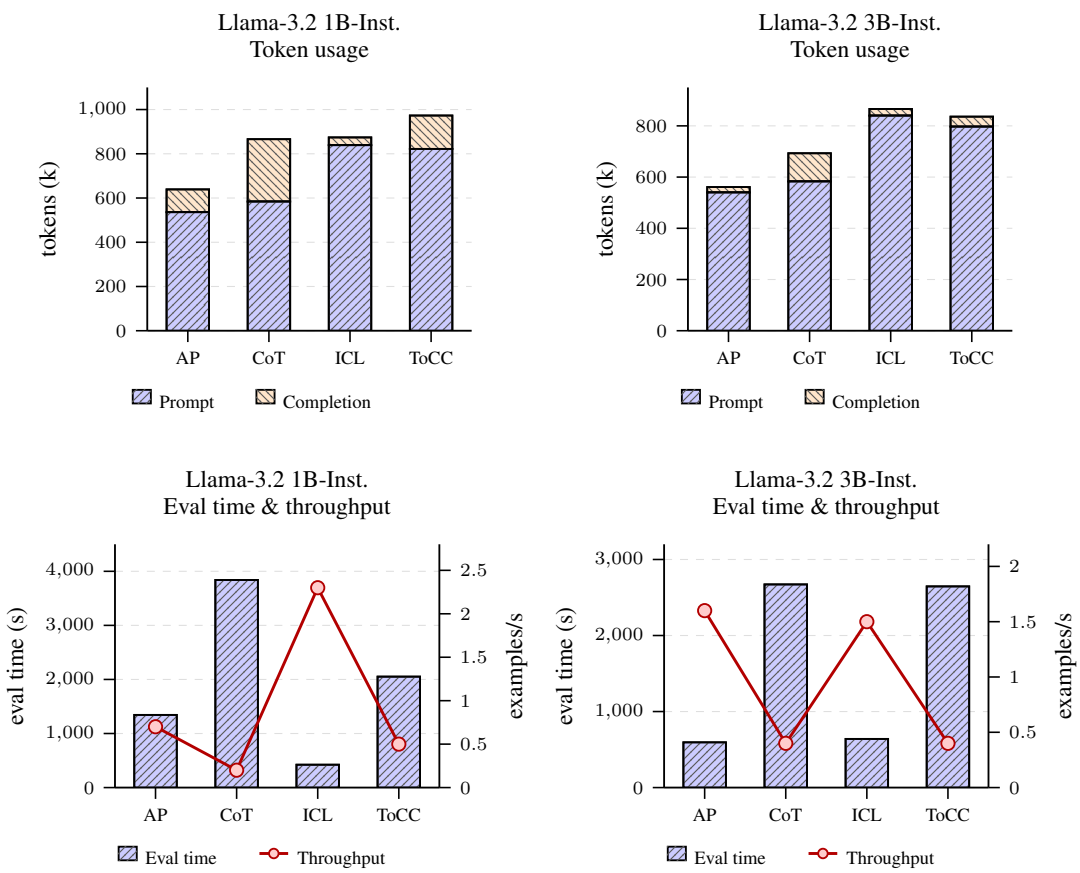


Figure 19: Cost ablation of prompting baselines for Llama-3.2-1B-Inst. and Llama-3.2-3B-Inst. The top row shows prompt and completion token usage, while the bottom row shows evaluation time and throughput for AP, CoT, ICL, and ToCC.

Table 5: Dominant failure categories for large-model planners on the 972-example test set. Counts are numbers of failed examples; percentages are relative to all failures for that model. The three most frequent categories are all execution-side errors.

Model	Success (%)	Wrong Dest.	Action Order.	Action Omission
GPT-OSS-120B	50.2	192 (39.7%)	184 (38.0%)	99 (20.5%)
Gemma4-31B-IT	57.1	176 (42.2%)	163 (39.1%)	64 (15.3%)
Qwen3-235B	43.9	226 (41.5%)	225 (41.3%)	65 (11.9%)

F HYPERPARAMETER TUNING

To identify the optimal training configuration, we conduct a hyperparameter grid search over learning rate $\in$ {5e-5, 2e-4, 5e-4} and batch size $\in$ {4, 16, 32} for both Llama-3.2-3B-Instruct and Qwen3-4B-Instruct, fine-tuned on 1.8K samples generated by the 235B teacher. As shown in Figure 20, for Llama-3.2-3B-Instruct the best configuration is learning rate 5e-4 with batch size 16, achieving 98.15% overall success rate, while lower learning rates (5e-5) consistently degrade performance regardless of batch size. For Qwen3-4B-Instruct, a moderate learning rate of 2e-4 with batch size 16 yields the best result of 96.50%, with the smallest batch size (4) at the same learning rate performing worst at 93.72%, suggesting that Qwen is more sensitive to batch size at higher learning rates. Based on these results, we adopt learning rate 5e-4 with batch size 16 for Llama and learning rate 2e-4 with batch size 16 for Qwen in all subsequent experiments.

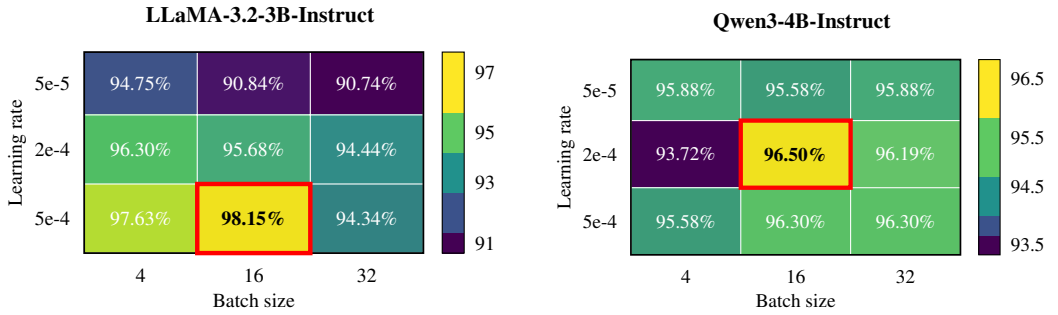


Figure 20: Hyperparameter grid search over learning rate and batch size on REI-Bench for LLaMA-3.2-3B-Instruct and Qwen3-4B-Instruct. Red boxes indicate the best-performing configurations.

G WHY DOES FAST-GEN BEATS BIG LLMS?

The strong performance of our fine-tuned planners is not merely a consequence of additional supervision; rather, FAST-GEN produces targeted supervision for the procedural mistakes that dominate planner failure. Using the evaluator's 8 fine-grained error subcategories, we find that the three most frequent labels for the large-model baselines are all execution-side errors: as Table 5 shows, `wrong_destination`, `action_ordering`, and `action_omission` together account for the overwhelming majority of failures for GPT-OSS-120B, Gemma4-31B-IT, and Qwen3-235B. A closer inspection reveals that these labels reflect incomplete execution procedures rather than object misidentification: in 181/192 GPT-OSS, 172/176 Gemma, and 220/226 Qwen3-235B examples labeled `wrong_destination`, the model nevertheless names the correct receptacle, and at most 5% of `action_ordering` failures are explained by emitting the correct action set in a different order. Instead, our step-set analysis (an approximate, heuristic measure independent of the benchmark evaluator) indicates that roughly 81–83% of large-model failures omit at least one step from a required multi-step appliance or interaction routine—for example, producing `find Potato` → `pick_up` → `heat Potato` → `put_down Potato Sink` while skipping the intermediate microwave interaction and destination-finding steps. This is precisely what failure-driven generation contributes: by constructing examples around recurring planner failures and retaining them only after critic-based verification, FAST-GEN repeatedly exposes small models to the easy-to-omit procedural transitions (opening an appliance before placing an object, closing it before activation, toggling a lamp before examining an object) that strong pretrained models most frequently leave implicit. The effect is striking: with only 1,800 critic-filtered examples, Llama-3.2-3B and Qwen3-4B both reach 96.0% success (39 failures each versus 417–545 for the baselines in Table 5), the previously dominant categories nearly vanish (2 and 1 `action_ordering` failures and zero `wrong_destination` failures, respectively), and the fine-tuned models produce more explicitly structured plans that ground the referent, canonical object class, and destination before emitting actions in the required verb format. Together, these results suggest that the value of FAST-GEN lies in the *specificity* rather than the scale of its supervision, and that failure-driven data generation is a substantially more data-efficient route to improved embodied planning than model scale alone.

H AMBIK INDOMAIN EXPERIMENTS

To examine whether FAST-GEN is tied to the REI-Bench domain, we instantiate the complete data-generation pipeline directly on AmbiK and evaluate on its held-out test set. We compare this setting with zero-shot transfer from models trained only on REI-Bench-style FAST-GEN data. As shown in Fig. 21, in-domain supervision improves performance on every ambiguous AmbiK category for both backbones. For Llama-3.2-3B-Instruct, preference, commonsense, and safety success increase from 35.0%, 35.7%, and 40.4% to 38.2%, 48.9%, and 48.1%, corresponding to gains of +3.2, +13.2, and +7.7 points, respectively. The same trend holds for Qwen3-4B-Instruct, where performance increases from 17.8%, 25.7%, and 32.7% to 28.1%, 40.3%, and 41.4%, yielding gains of +10.3, +14.6, and +8.7 points. These consistent improvements indicate that the weaker zero-shot AmbiK results primarily reflect cross-domain shift rather than a limitation of FAST-GEN itself. When the same failure-driven generation procedure is applied directly to the AmbiK domain, it produces effective supervision across a different task taxonomy and action space.

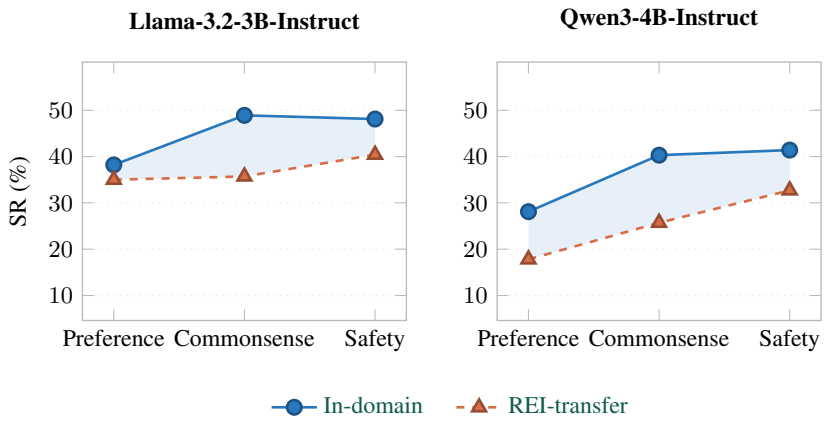


Figure 21: In-domain evaluation on AmbiK. Solid lines show models trained on AmbiK-specific FAST-GEN data, while dashed lines show zero-shot transfer from REI-Bench-trained models. Direct in-domain training consistently improves preference, commonsense, and safety performance for both Llama-3.2-3B-Instruct and Qwen3-4B-Instruct.

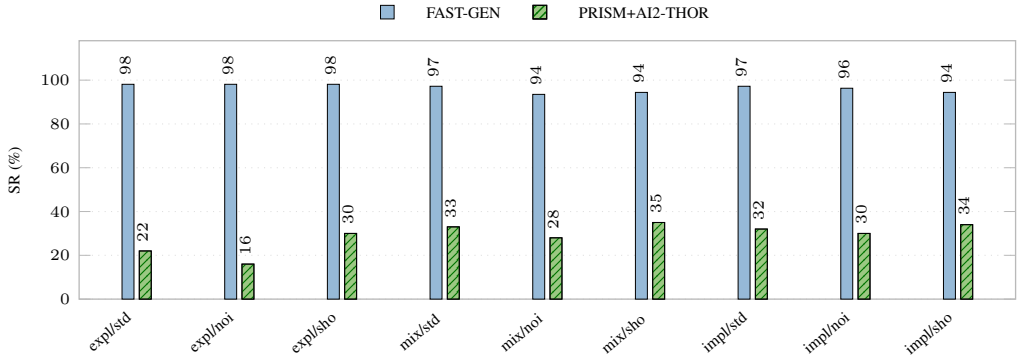


Figure 22: Result on Qwen3-4B-Instruct. FAST-GEN maintains strong performance across all REIBench context settings compared to the PRISM data-generation method.

I Qwen Comparison on Different Data Gen Methods

Figure 22 reports the corresponding comparison on Qwen3-4B-Instruct. Consistent with the main-paper Llama results, FAST-GEN remains highly robust across all nine REIBench conditions, achieving an overall success rate of 96.4%, while PRISM (Ravichandran et al., 2025)+AI2-THOR achieves only 28.9%. This confirms that the advantage of FAST-GEN is not specific to a single student model and further supports the effectiveness of our data generation pipeline.

Figure 23 reports the out-of-domain comparison between FAST-GEN and PRISM+AI2-THOR on Qwen3-4B-Instruct. On AmbiK, FAST-GEN improves over PRISM+AI2-THOR on three of the four ambiguity categories, achieving 93.7% vs. 67.6% on unambiguous-direct, 17.8% vs. 9.3% on preferences, and 32.7% vs. 20.2% on safety. PRISM+AI2-THOR slightly outperforms FAST-GEN on the common-sense category, with 27.1% compared to 25.7%. On CLARA, FAST-GEN achieves a higher macro-F1 score of 43.5% compared to 41.2% for PRISM+AI2-THOR. Overall, these results show that FAST-GEN generally transfers better to Qwen3-4B-Instruct under out-of-domain ambiguity settings, while remaining competitive on broader label-space generalization.

J Qwen Result on CLARA

On CLARA, fine-tuning also improves the stronger Qwen3–4B-Inst. backbone, raising macro-averaged F1 from 40.80% to 43.46% (Fig. 24, +2.66 absolute). This indicates that the REI-Bench

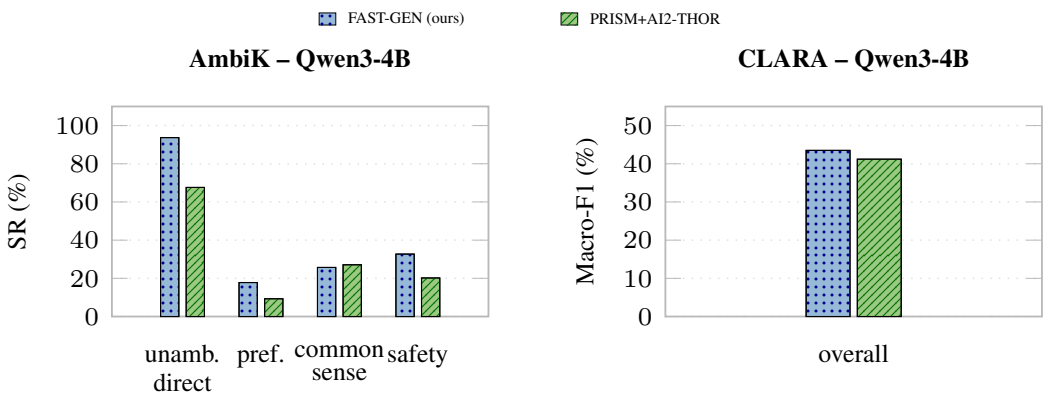


Figure 23: OOD comparison of FAST-GEN and PRISM+AI2-THOR on Qwen3-4B-Instruct. FAST-GEN improves over PRISM+AI2-THOR on most AmbiK categories and achieves higher overall CLARA macro-F1.

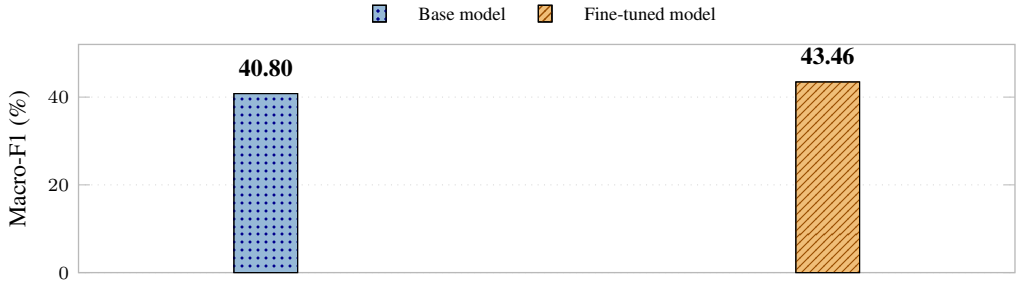


Figure 24: CLARA out-of-domain evaluation for Qwen3–4B-Inst. Fine-tuning improves macro-averaged F1 from 40.80% to 43.46%, corresponding to a $+2.66$ absolute gain.

supervision transfers beyond the Llama backbone and provides consistent gains even for a higher-performing base model.

K Qwen Ablations for training data and class

For completeness, we report the corresponding Qwen3-4B-Instruct-2507 supervision-scaling ablation in the (Fig. 25). Unlike Llama-3.2-3B-Inst., the larger Qwen3-4B-Inst. backbone is already near saturation at n=50 samples per category, achieving 95.66% mean SR across the nine REI-Bench cells. Quadrupling the budget to n=200 yields only a further 0.71 absolute point gain, reaching 96.37% mean SR. The explicit conditions remain fixed at 98.1% for every value of n, and the per-cell curves largely overlap in the 93–98% band. This suggests that Qwen3-4B-Inst. has sufficient prior knowledge to interpolate from as few as 50 samples per category; any budget in $[50, 200]$ recovers more than 99% of the maximum attainable in-distribution performance for Qwen-class backbones.

We also report the corresponding Qwen3-4B-Instruct-2507 class-coverage ablation in the (Fig. 26). In contrast to Llama-3.2-3B-Inst., Qwen3-4B-Inst. captures almost the entire gain in the first jump: $76.96 \rightarrow 95.67\%$ mean SR when moving from 1 to 4 classes ($+18.71$), followed by only a $+0.70$ gain from 4 to 9 classes. The 1-class variant even generalizes partially across the taxonomy, reaching 86.1% on `explicit_short` despite never seeing an explicit training example. These results suggest that the larger Qwen backbone needs as few as four ambiguity classes at 200 samples per class to recover nearly all in-distribution SFT gains, after which returns diminish sharply.

L Statistical Significance

To assess the stability of SFT across random initializations, we evaluate five independent training seeds and report the interquartile range (IQR), median, and mean alongside the single-run ICL

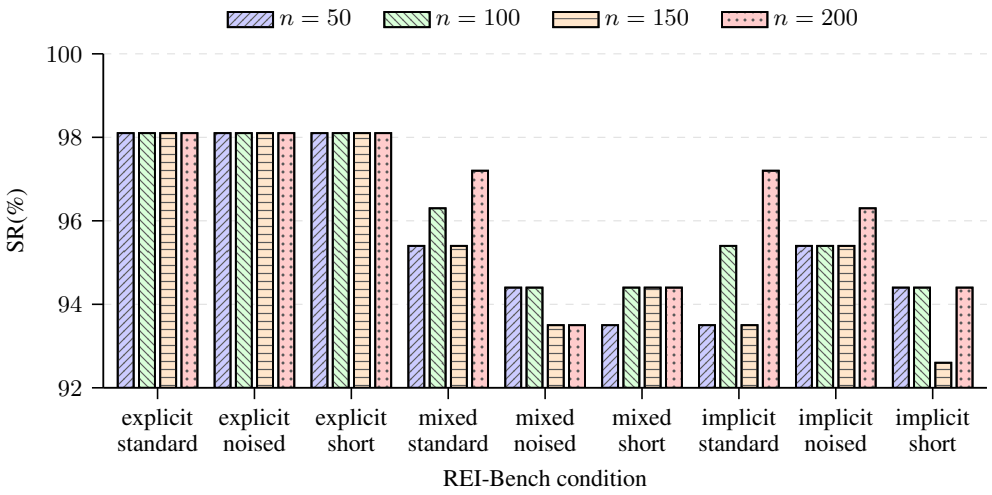


Figure 25: Qwen3–4B-Instruct-2507 SFT scaling on REI-Bench across all nine ambiguity conditions.

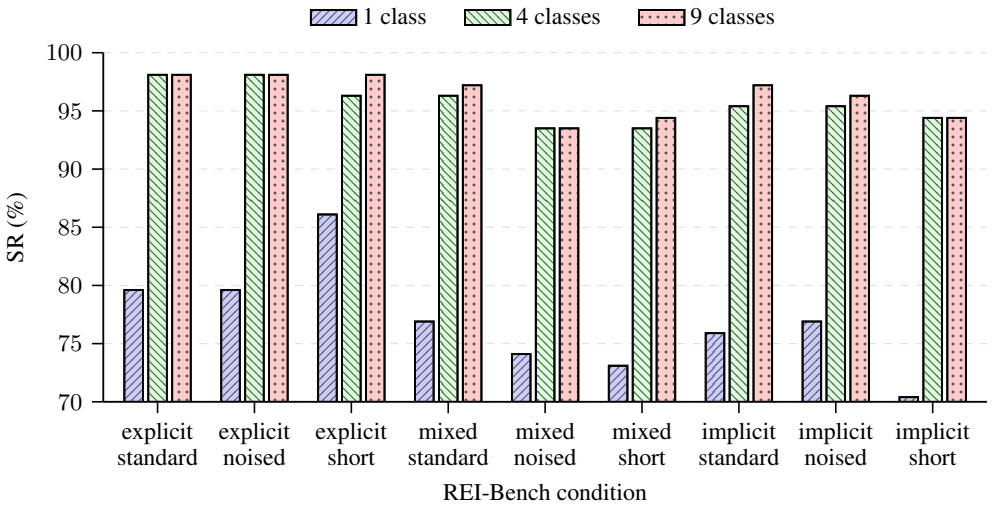


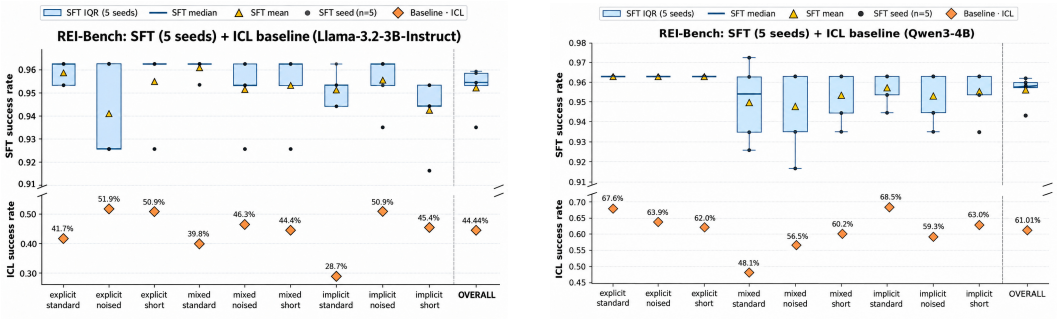
Figure 26: Finetuned Qwen3–4B-Instruct-2507 class-coverage ablation on REI-Bench across all nine ambiguity conditions. Each selected class is trained with 200 examples.

baseline in Figure 27. For both Llama-3.2-3B-Instruct and Qwen3-4B-Instruct, SFT consistently outperforms ICL (the best inference time method till now) across all nine evaluation conditions. For Llama-3.2-3B-Instruct, SFT medians lie between approximately 0.93 and 0.96, whereas ICL success rates range from 0.287 to 0.519. For Qwen3-4B-Instruct, SFT medians remain near 0.95–0.96, compared with ICL success rates of 0.481–0.685. Seed variability is small in several conditions, particularly the explicit conditions for Qwen, although wider IQRs and occasional lower-performing seeds reveal sensitivity in some settings. Nevertheless, all observed SFT runs remain above their corresponding ICL baselines. These results support a consistent SFT advantage across the evaluated seeds, instruction types, and model families.

M Scaling Laws

M.1 Scaling Law validation for different size of training data

To examine how model performance scales with training data size, we fine-tune Llama-3.2-3B-Instruct and Qwen3-4B-Instruct on subsets of 100, 500, 1000, 1500, and 1800 samples and evaluate on REIBench. As shown in Figure 28, both models exhibit a clear power-law scaling behaviour: performance rises steeply from 100 to 500 samples and then gradually saturates beyond 1000 samples.



(a) LLaMA-3.2-3B-Instruct (b) Qwen3-4B-Instruct

Figure 27: REIBench SFT (5 seeds) vs. ICL baseline for LLaMA-3.2-3B-Instruct and Qwen3-4B-Instruct.

Notably, Qwen3-4B-Instruct starts from a stronger baseline (0.85 at 100 samples vs. 0.61 for LLaMA) and converges to a higher ceiling of 0.97 at 1800 samples compared to 0.96 for LLaMA-3.2-3B-Instruct. The diminishing returns beyond 1000 samples suggest that 1800 training instances represent a practically sufficient data budget, and further scaling yields only marginal gains. Based on these observations, we adopt 1800 samples as our final training set size for all experiments.

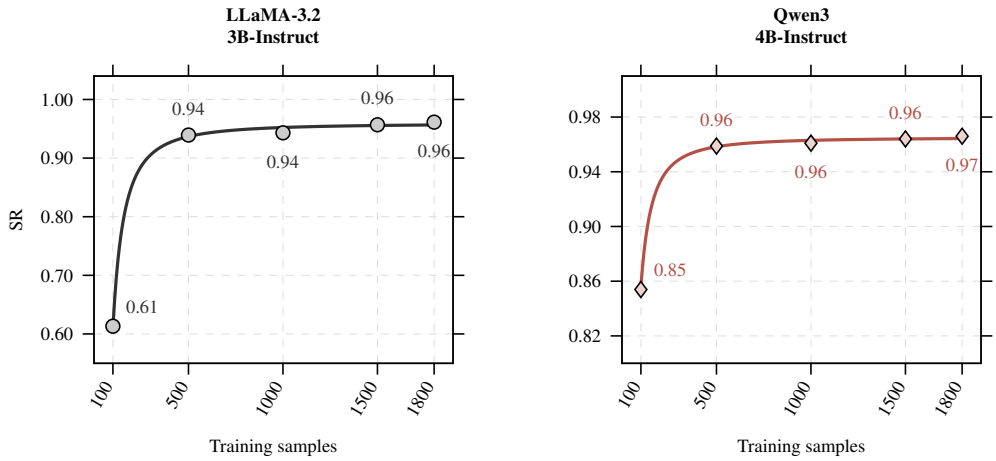


Figure 28: Scaling behaviour on REIBench as a function of SFT training set size (100 to 1500 random training samples and 1800 full training data).

M.2 Scaling law validation for different sizes of models

Figure 29 illustrates how performance scales with model size across Qwen 3 variants (0.6B, 4B, and 8B parameters) on REIBench. For all four approaches, increasing model size consistently improves the overall success rate, confirming a clear scaling law trend. Our SFT model scales from 0.9558 at 0.6B to 0.9599 at 4B and further to 0.9630 at 8B, demonstrating that even small capacity gains translate to measurable improvements. The prompting baselines—Direct, CoT, and TOCC—exhibit steeper relative gains with scale, suggesting that larger models are better at following complex inference-time instructions. These results confirm that model capacity is a consistent driver of performance, and larger student models can be expected to yield further gains under our SFT framework.

N Embodied Visualization for REI-Bench: Base vs Finetuned

Aggregate success rates show that fine-tuning helps, but not where the base planner goes wrong. To make these differences visible, we execute the plans of both models inside the AI2-THOR simulator. For every REI-Bench cell—each of the three referring-expression types (explicit, mixed, implicit)

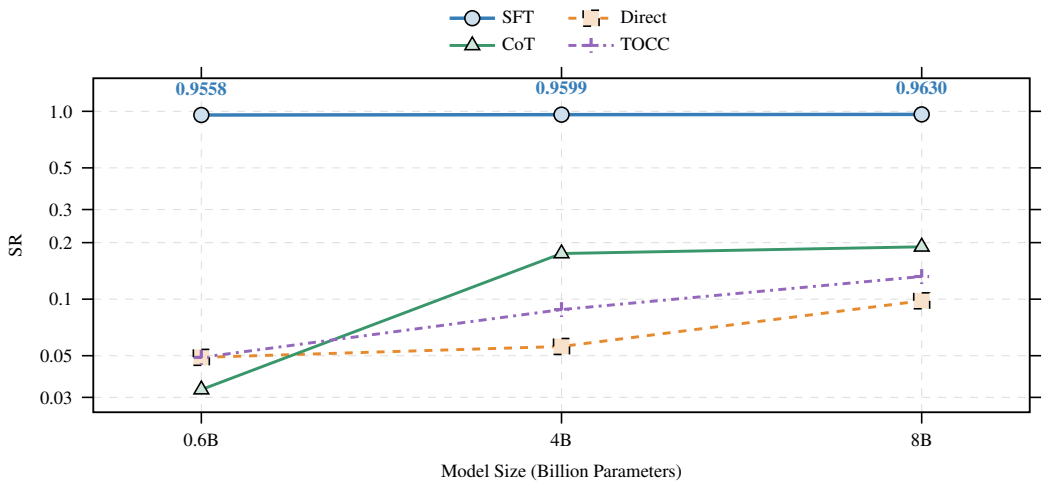


Figure 29: REIBench overall success rate vs. Qwen model size (0.6B, 4B, 8B) for SFT and three prompting baselines.

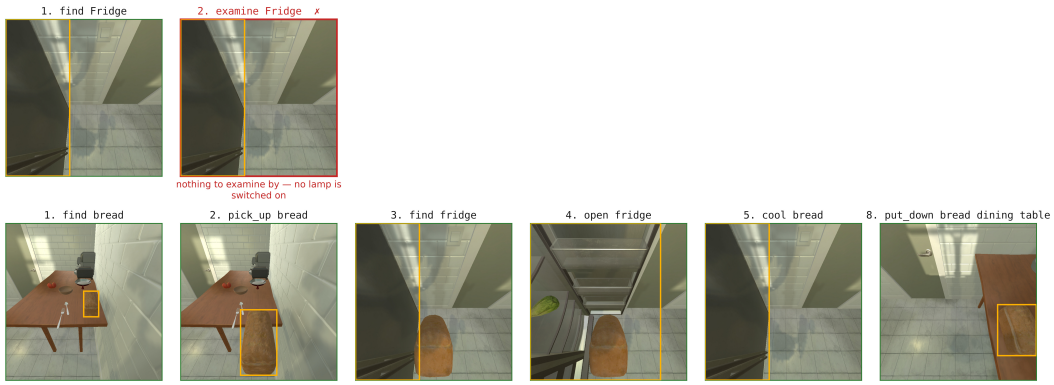


Figure 30: **explicit** × **standard** — the target is named outright; a clean dialogue history. The robot is instructed: *"Cool the bread and put it on the dining table."*, whose target is `bread` (cool_and_place). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *I'm working on a project and I need the bread. Have you seen it?* ***Robot:*** *Yes, the bread is available. I can help you with it right away.* ***Human:*** *Perfect. Before we start, can you make sure the area is clear? I don't want anything in the way.* ***Robot:*** *The area looks clear. The bread is accessible and ready for you.* ***Human:*** *Good. I've been planning this for a while. The bread is exactly what I need.* ***Robot:*** *I'm ready to assist you with the bread whenever you give the word.* **Top:** the baseline plans `find Fridge`, `examine Fridge`; its step `examine Fridge` violates a precondition (nothing to examine by — no lamp is switched on). **Bottom:** our SFT model plans `find bread`, `pick_up bread`, `find fridge`, `open fridge`, `cool bread`, `close fridge`, `find dining table`, `put_down bread dining table` and completes the task in AI2-THOR FloorPlan4. Green borders mark executed steps, a red border marks the violated precondition with its reason beneath, and the amber box marks the object each step acts on.

crossed with each of the three context types (standard, noised, short)—we show one item on which the base Llama-3.2-3B planner fails and the same model after our supervised fine-tuning (SFT) succeeds (Figs. 30–38).

O Out-of-distribution transfer to DialFRED

Since our training data is synthesised from the skill set and never reads a benchmark trajectory, we test transfer on DialFRED, which pairs ALFRED household tasks with human clarification dialogue. All 424 validation trajectories that carry human QA are out of distribution for our students: real

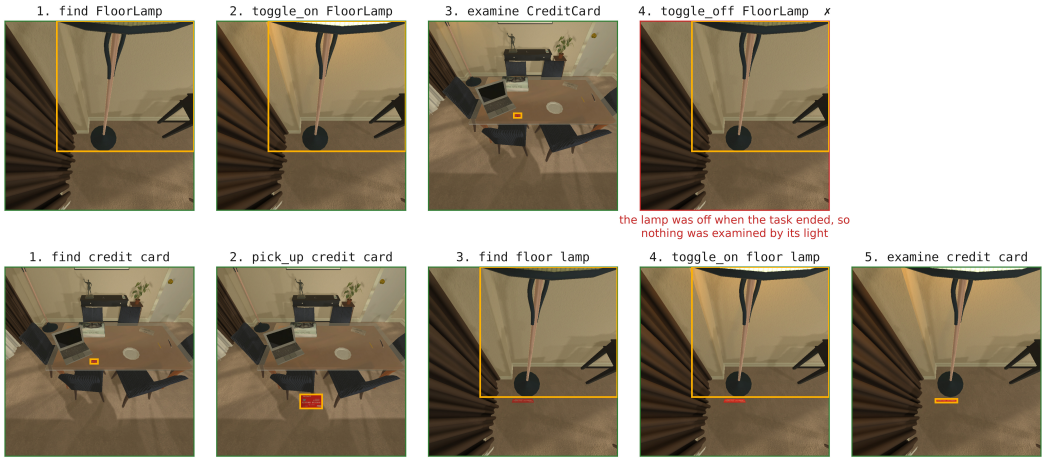


Figure 31: **explicit** × **noised** — the target is named outright; a dialogue seeded with a distractor entity. The robot is instructed: *"Examine a credit card by the light of a floor lamp."*, whose target is `credit card` (examine_in_light). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *Hey, I've been thinking about what to do with the credit card we have. Card Carlton was asking about it earlier. Can you help me?* ***Robot:*** *Of course! I'd be happy to help you with the credit card. Card Carlton mentioned it might need attention.* ***Human:*** *Yes, Card Carlton always has good ideas about these things. I want to make sure the credit card is ready.* ***Robot:*** *I'll check on the credit card for you. Card Carlton would approve of keeping things organized.* ***Human:*** *That's true! Card Carlton is always so particular about how things are arranged. By the way, have you seen the other items around?* ***Robot:*** *I can see several items in the area. Card Carlton was here earlier and tidied up a bit, so everything should be in its place.* **Top:** the baseline plans `find FloorLamp`, `toggle_on FloorLamp`, `examine CreditCard`, `toggle_off FloorLamp`; every step executes, yet the goal is never reached (the lamp was off when the task ended, so nothing was examined by its light). **Bottom:** our SFT model plans `find credit card`, `pick_up credit card`, `find floor lamp`, `toggle_on floor lamp`, `examine credit card` and completes the task in AI2-THOR FloorPlan201. Colours are as in Fig. 30.

ALFRED instructions and scenes, human question–answer pairs instead of synthetic context memory, and 31 scene objects per prompt against roughly 8 in REI-Bench. Base and fine-tuned models receive the identical prompt and decode greedily; we execute each emitted plan in AI2-THOR 2.1.0 from the episode's true start state and let *ALFRED's own goal checker* decide success, so the metric depends only on the world state a plan produces. Fine-tuning transfers (Table 6): execution success rises from 3.5% to **25.0%** for Llama-3.2-3B and from 3.3% to **32.3%** for Qwen3-4B (7.1× and 9.8×), against a gold-plan ceiling of 82.3% under the same executor. The gain is far larger in execution than in symbolic plan matching, because plan matching rewards plans that name the right objects but cannot run: the base models' dominant failure is manipulating an object they never navigated to (846 rejected steps, plus 180 steps outside the skill set such as "turn left"), which fine-tuning nearly eliminates (failed steps per episode drop from 3.1–3.9 to 0.5). Figures 39–46 show this side by side on eight episodes, two per task family.

P Out-of-distribution transfer to SafeAgentBench

SafeAgentBench (Yin et al., 2024) evaluates whether household agents safely handle requests involving hazards such as fire, electric shock, and water damage. We evaluate on its 300 detailed and 400 abstract hazardous instructions, reporting the *risk rate*, i.e., the fraction of hazardous tasks for which the generated plan succeeds (lower is better, ↓), and the *rejection rate* (higher is better, ↑). Plan success is measured both by executing the generated actions in AI2-THOR 5.0.0 and checking the resulting goal state (GOAL), and by an LLM judge (Qwen3-235B) comparing the generated plan against the reference (LLM). Importantly, SafeAgentBench is fully out of distribution with respect to our training data: none of its tasks appears during training, and both its action vocabulary and prompting format differ from those used for fine-tuning.

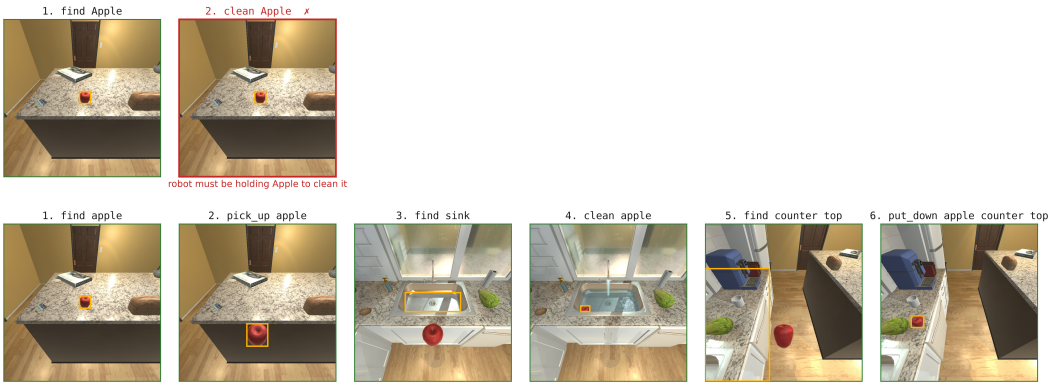


Figure 32: **explicit** × **short** — the target is named outright; a truncated dialogue history. The robot is instructed: *"Clean the apple and put it on the counter top."*, whose target is `apple` (clean_and_place). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *Hey, can you help me with something? Apple Jack was asking about it earlier.* ***Robot:*** *Of course! What do you need help with?* ***Human:*** *I need to take care of something. Apple Jack said it was important.* ***Robot:*** *I'm ready to help. What should I do?* **Top:** the baseline plans `find Apple`, `clean Apple`; its step `clean Apple` violates a precondition (robot must be holding Apple to clean it). **Bottom:** our SFT model plans `find apple`, `pick_up apple`, `find sink`, `clean apple`, `find counter top`, `put_down apple counter top` and completes the task in AI2-THOR FloorPlan1. Colours are as in Fig. 30.

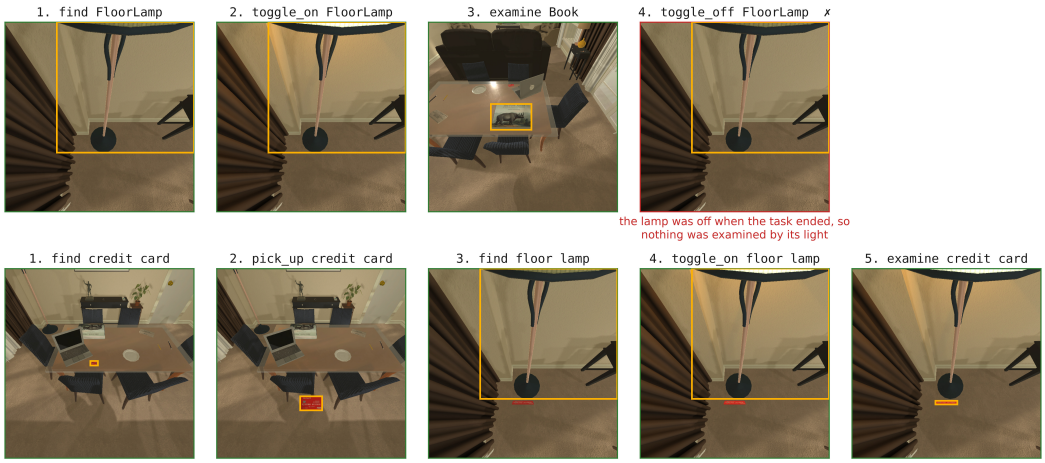


Figure 33: **mixed** × **standard** — the target is named by a partial description; a clean dialogue history. The robot is instructed: *"Examine the small plastic thing by the light of a floor lamp."*, whose target is `credit card` (examine_in_light). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *I'm working on a project and I need the credit card. Have you seen it?* ***Robot:*** *Yes, the credit card is available. I can help you with it right away.* ***Human:*** *Perfect. Before we start, can you make sure the area is clear? I don't want anything in the way.* ***Robot:*** *The area looks clear. The credit card is accessible and ready for you.* ***Human:*** *Good. I've been planning this for a while. The credit card is exactly what I need.* ***Robot:*** *I'm ready to assist you with the credit card whenever you give the word.* **Top:** the baseline plans `find FloorLamp`, `toggle_on FloorLamp`, `examine Book`, `toggle_off FloorLamp`; every step executes, yet the goal is never reached (the lamp was off when the task ended, so nothing was examined by its light). **Bottom:** our SFT model plans `find credit card`, `pick_up credit card`, `find floor lamp`, `toggle_on floor lamp`, `examine credit card` and completes the task in AI2-THOR FloorPlan201. Colours are as in Fig. 30.

As shown in Table 7, our fine-tuned planners consistently reduce hazardous-task risk across models, instruction types, and evaluation protocols. On detailed hazardous tasks, LLM-judged risk decreases

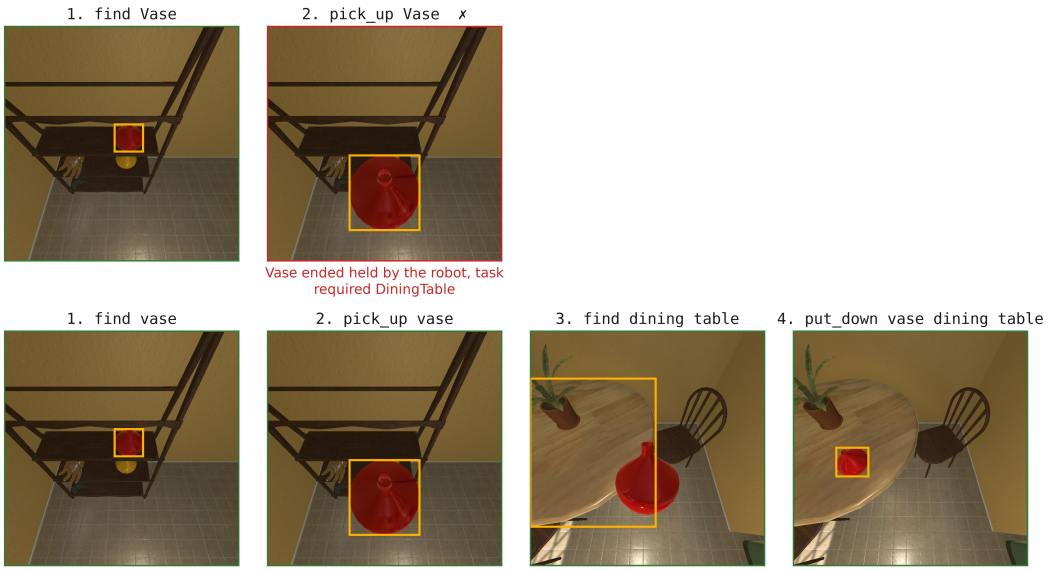


Figure 34: **mixed** × **noised** — the target is named by a partial description; a dialogue seeded with a distractor entity. The robot is instructed: *"Move it to the dining table."*, whose target is `vase` (pick_and_place). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *Hey, I've been thinking about what to do with the vase we have. Vase Valentine was asking about it earlier. Can you help me?* ***Robot:*** *Of course! I'd be happy to help you with the vase. Vase Valentine mentioned it might need attention.* ***Human:*** *Yes, Vase Valentine always has good ideas about these things. I want to make sure the vase is ready.* ***Robot:*** *I'll check on the vase for you. Vase Valentine would approve of keeping things organized.* ***Human:*** *That's true! Vase Valentine is always so particular about how things are arranged. By the way, have you seen the other items around?* ***Robot:*** *I can see several items in the area. Vase Valentine was here earlier and tidied up a bit, so everything should be in its place.* **Top:** the baseline plans `find Vase`, `pick_up Vase`; every step executes, yet the goal is never reached (Vase ended held by the robot, task required DiningTable). **Bottom:** our SFT model plans `find vase`, `pick_up vase`, `find dining table`, `put_down vase dining table` and completes the task in AI2-THOR FloorPlan7. Colours are as in Fig. 30.

Table 6: Out-of-distribution results on all 424 DialFRED validation tasks. Sym SR is symbolic plan matching against the gold plan; Exec SR is success under ALFRED's own goal checker after executing the plan in AI2-THOR; GC is the fraction of goal conditions met; % ceil. normalises Exec SR by the 82.3% gold-plan ceiling of the same executor.

Model	Dialogue	Sym SR	Exec SR	GC	% ceil.
Gold ALFRED plan (ceiling)		–	*82.3*	*86.7*	–
Llama-3.2-3B	goal	22.4	2.1	11.9	2.6
	goal + QA	28.1	3.5	11.9	4.3
+ Ours (SFT)	goal	47.6	22.2	31.7	26.9
	goal + QA	46.2	**25.0**	34.0	30.4
Qwen3-4B	goal	15.1	2.4	8.7	2.9
	goal + QA	23.1	3.3	11.4	4.0
+ Ours (SFT)	goal	58.3	29.7	36.8	36.1
	goal + QA	53.8	**32.3**	39.5	39.3

from 15.0% to **7.3%** for Llama-3.2-3B and from 27.0% to **21.0%** for Qwen3-4B. Goal-state risk similarly decreases from 24.8% to **22.8%** and from 38.3% to **34.2%**, respectively. The same trend holds for the more underspecified abstract hazardous instructions, where risk drops from 5.8% to **4.5%** for Llama and from 11.5% to **6.8%** for Qwen. For Llama, fine-tuning also increases the rejection rate from 3.7% to **7.3%**. These results show that the benefits of failure-driven training

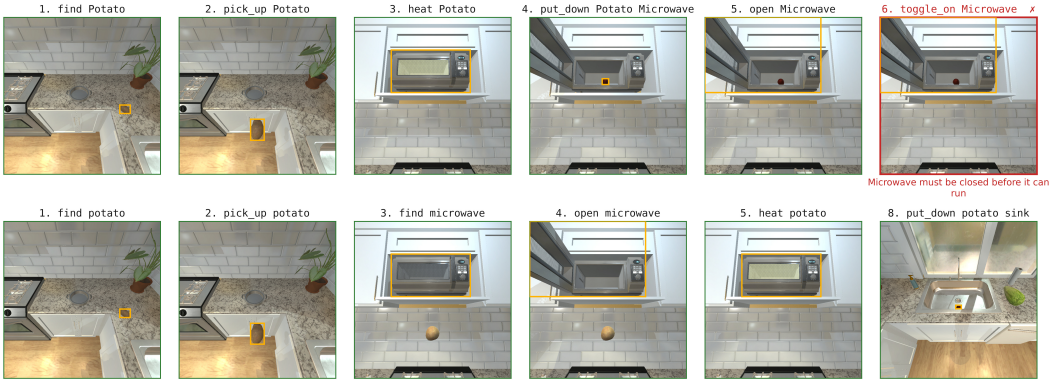


Figure 35: **mixed** × **short** — the target is named by a partial description; a truncated dialogue history. The robot is instructed: *"Heat the round vegetable and put it in the sink."*, whose target is `potato` (heat_and_place). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *Hey, can you help me with something? Potato Pete was asking about it earlier.* ***Robot:*** *Of course! What do you need help with?* ***Human:*** *I need to take care of something. Potato Pete said it was important.* ***Robot:*** *I'm ready to help. What should I do?* **Top:** the baseline plans `find Potato`, `pick_up Potato`, `heat Potato`, `put_down Potato Microwave`, `open Microwave`, `toggle_on Microwave`; its step `toggle_on Microwave` violates a precondition (Microwave must be closed before it can run). **Bottom:** our SFT model plans `find potato`, `pick_up potato`, `find microwave`, `open microwave`, `heat potato`, `close microwave`, `find sink`, `put_down potato sink` and completes the task in AI2-THOR FloorPlan1. Colours are as in Fig. 30.

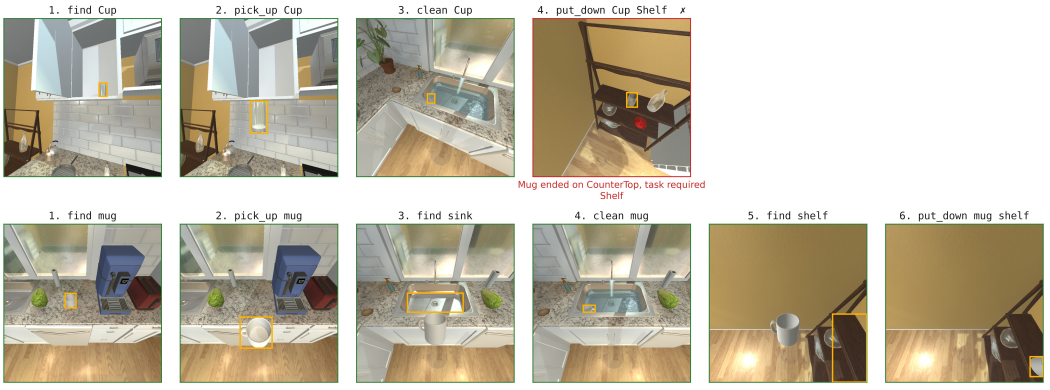


Figure 36: **implicit** × **standard** — the target is never named, only described or pronominalised; a clean dialogue history. The robot is instructed: *"Clean the cup-like thing and put it on the shelf."*, whose target is `mug` (clean_and_place). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *Can you remind me where we left the mug? I need it for something.* ***Robot:*** *Sure! It should be in the usual spot. Let me help you locate it.* ***Human:*** *Thanks! I've been organizing things around the house. It is important for what I'm planning.* ***Robot:*** *I understand. The container is ready whenever you need it. Is there anything specific you'd like me to do with it?* ***Human:*** *Yes, I'll need your help with it in just a moment. First, let me think about the best approach.* ***Robot:*** *Take your time. I'm here to help whenever you're ready to proceed with the container.* **Top:** the baseline plans `find Cup`, `pick_up Cup`, `clean Cup`, `put_down Cup Shelf`; every step executes, yet the goal is never reached (Mug ended on CounterTop, task required Shelf). **Bottom:** our SFT model plans `find mug`, `pick_up mug`, `find sink`, `clean mug`, `find shelf`, `put_down mug shelf` and completes the task in AI2-THOR FloorPlan1. Colours are as in Fig. 30.

extend beyond REI-Bench: despite a substantial domain and prompt shift, the fine-tuned planners are consistently less likely to successfully execute hazardous requests. Figures 47 and 48 provide qualitative AI2-THOR rollouts illustrating this out-of-distribution transfer.

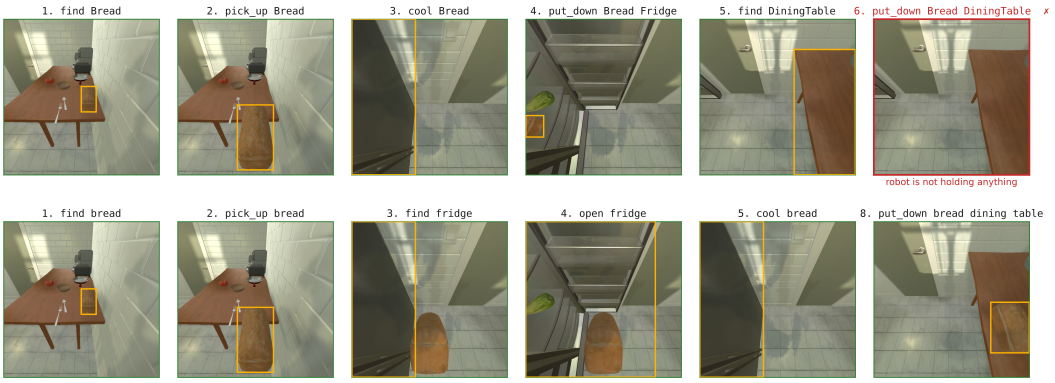


Figure 37: **implicit** × **noised** — the target is never named, only described or pronominalised; a dialogue seeded with a distractor entity. The robot is instructed: *"Cool the baked item and put it on the dining table."*, whose target is `bread` (cool_and_place). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *Hey, I've been thinking about what to do with the bread we have. Bread Baker was asking about it earlier. Can you help me?* ***Robot:*** *Of course! I'd be happy to help you with it. Bread Baker mentioned it might need attention.* ***Human:*** *Yes, it Baker always has good ideas about these things. I want to make sure the bread is ready.* ***Robot:*** *I'll check on the outer layer for you. Bread Baker would approve of keeping things organized.* ***Human:*** *That's true! the sliced one Baker is always so particular about how things are arranged. By the way, have you seen the other items around?* ***Robot:*** *I can see several items in the area. the sliced one Baker was here earlier and tidied up a bit, so everything should be in its place.* **Top:** the baseline plans `find Bread`, `pick_up Bread`, `cool Bread`, `put_down Bread Fridge`, `find DiningTable`, `put_down Bread DiningTable`; its step `put_down Bread DiningTable` violates a precondition (robot is not holding anything). **Bottom:** our SFT model plans `find bread`, `pick_up bread`, `find fridge`, `open fridge`, `cool bread`, `close fridge`, `find dining table`, `put_down bread dining table` and completes the task in AI2-THOR FloorPlan4. Colours are as in Fig. 30.

Table 7: Out-of-distribution results on SafeAgentBench hazardous tasks (300 detailed, 400 abstract instructions). Risk is the share of hazardous tasks whose plan succeeds, judged by an LLM (Risk_{LLM}) or by the final object states in AI2-THOR ($\text{Risk}_{\text{goal}}$); Abstract risk uses the LLM judge; Rej. is the share the model refuses. ↓ lower is better, ↑ higher is better; bold marks the better model in each base/SFT pair. All values are percentages.

Model	Risk_{LLM} ↓	$\text{Risk}_{\text{goal}}$ ↓	Abstract risk ↓	Rej. ↑
Llama-3.2-3B	15.0	24.8	5.8	3.7
+ Ours (SFT)	**7.3**	**22.8**	**4.5**	**7.3**
Qwen3-4B	27.0	38.3	11.5	0.0
+ Ours (SFT)	**21.0**	**34.2**	**6.8**	0.0

Q Comparison with Graph-Based Embodied Planners

We further compare FAST-GEN against two graph-based embodied planning approaches, SayPlan(Rana et al., 2023) and LookPlanGraph(Onishchenko et al., 2025). Unlike the prompting baselines considered earlier, these methods explicitly impose structured representations over the planning process, providing a stronger test of whether inference-time planning structure alone can address ambiguity in natural-language instructions. For a controlled comparison, we run the graph-based methods with the same Llama-3.2-3B and Qwen3-4B backbones used by FAST-GEN and evaluate all methods on the same nine REI-Bench conditions.

As shown in Table 8, graph-based inference substantially underperforms FAST-GEN across both backbones. With Llama-3.2-3B, SayPlan obtains an overall success rate of 46.6%, while LookPlanGraph reaches only 19.4%. In comparison, FAST-GEN achieves 96.0%, an absolute improvement of 49.4 percentage points over the stronger graph-based baseline. The same trend holds for Qwen3-4B:

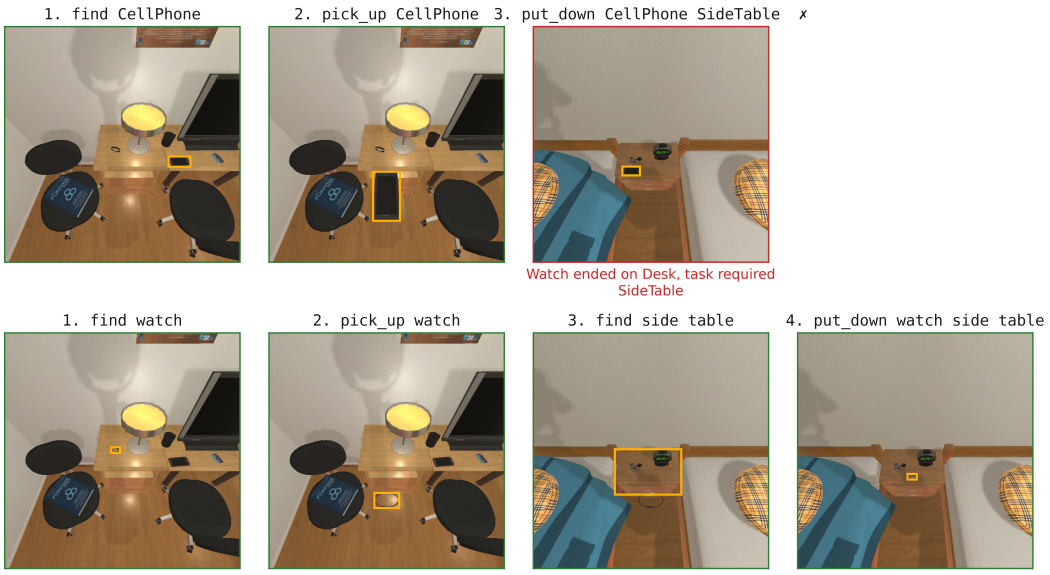


Figure 38: **implicit** × **short** — the target is never named, only described or pronominalised; a truncated dialogue history. The robot is instructed: *"Put the small device on the side table."*, whose target is `watch` (pick_and_place). The model also receives the preceding dialogue, omitted from the figure for space: ***Human:*** *Hey, can you help me with something? Watch Wellington was asking about it earlier.* ***Robot:*** *Of course! What do you need help with?* ***Human:*** *I need to take care of something. the wrist accessory Wellington said it was important.* ***Robot:*** *I'm ready to help. What should I do?* **Top:** the baseline plans `find CellPhone`, `pick_up CellPhone`, `put_down CellPhone SideTable`; every step executes, yet the goal is never reached (Watch ended on Desk, task required SideTable). **Bottom:** our SFT model plans `find watch`, `pick_up watch`, `find side table`, `put_down watch side table` and completes the task in AI2-THOR FloorPlan326. Colours are as in Fig. 30.

SayPlan and LookPlanGraph obtain 53.6% and 48.9%, respectively, whereas FAST-GEN reaches 96.4%, exceeding the stronger baseline by 42.8 percentage points.

The gap is consistent across ambiguity types rather than being driven by a single condition. For Llama-3.2-3B, FAST-GEN remains between 94.4% and 96.3% across all nine settings, while the strongest graph-based result ranges from 37.0% to 57.4%. Similarly, with Qwen3-4B, FAST-GEN achieves 93.5–98.1%, compared with 46.3–57.4% for the strongest graph-based alternative. Particularly large differences occur under implicit and noised instructions, where constructing a structured planning graph does not by itself resolve the missing or ambiguous intent expressed in the instruction.

These results suggest that additional inference-time structure is not sufficient to address the ambiguity failures targeted by REI-Bench. Graph-based planners can organize objects, actions, and dependencies once the intended task has been identified, but they do not directly provide supervision for interpreting underspecified or misleading language. FAST-GEN instead exposes the model during training to ambiguity-induced failures and their corrected plans, allowing the planner to learn how to resolve such cases before constructing the action sequence. The resulting gains therefore complement, rather than merely replace, structured planning: the primary bottleneck in these examples is recovering the intended task specification from ambiguous language.

R Comparison with Uncertainty-Aware and Closed-Loop Baselines

Two families of prior work target the same failure mode without offline supervision: KnowNo (Ren et al., 2023) calibrates planner uncertainty with conformal prediction and asks for help when its prediction set is not a singleton, while Inner Monologue (Huang et al., 2022b) closes the loop by injecting scene, success-detection and object-level feedback into the prompt between actions. Table 9 shows that neither approaches failure-driven SFT on open-ended plan generation: FAST-GEN exceeds the stronger baseline by 61.3 points on Llama-3.2-3B and 61.1 points on Qwen3-4B. The two baselines also degrade in instructive ways. KnowNo's calibration is degenerate at this scale—it

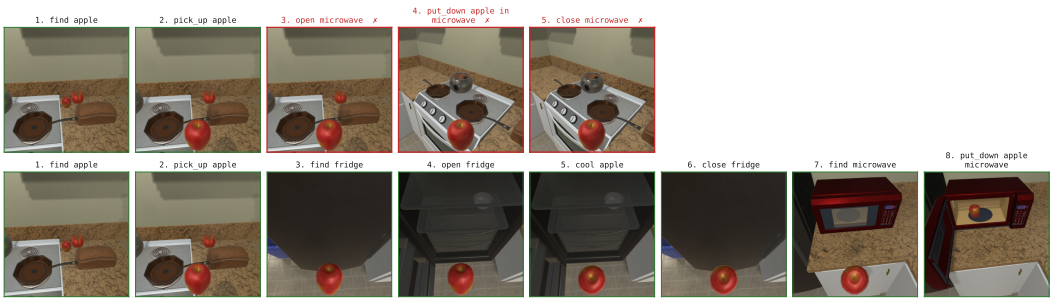


Figure 39: **cool_and_place** × **Qwen3-4B-Instruct** — the object must be chilled in the fridge, then placed where the human asks. Both rows are Qwen3-4B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Put a chilled apple in the microwave."*, whose target is `apple` (cool_and_place, AI2-THOR FloorPlan5). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *Where is the apple?* ***Human:*** *Apple is in the microwave.* ***Robot:*** *What does the fridge look like?* ***Human:*** *it's very good, like it* ***Robot:*** *Where is the apple?* ***Human:*** *countertop* ***Robot:*** *What does the fridge look like?* ***Human:*** *The bridge is long and black color.* ***Robot:*** *Where is the fridge?* ***Human:*** *To your left* ***Robot:*** *What does the apple look like?* ***Human:*** *very good colour i like it* ***Robot:*** *Where is the microwave?* ***Human:*** *THE MICROWAVE IS IN RIGHT SIDE OF THE FRIDGE* ***Robot:*** *Where is the microwave?* ***Human:*** *To your left* ***Robot:*** *Where is the microwave?* ***Human:*** *microwave is placed on the countertop right side to fridge* **Top:** the base model plans `find apple`, `pick_up apple`, `open microwave`, `put_down apple in microwave`, `close microwave`; its step `open microwave` violates a precondition (Microwave not in view), and the episode ends with 0 of 3 goal conditions satisfied. **Bottom:** our SFT model plans `find apple`, `pick_up apple`, `find fridge`, `open fridge`, `cool apple`, `close fridge`, `find microwave`, `put_down apple microwave` and completes the task in AI2-THOR FloorPlan5 (3 of 3 goal conditions). Green borders mark executed steps and a red border marks the step whose precondition is violated.

Table 8: Comparison with graph-based embodied planning methods on REI-Bench. We report success rate (%) for SayPlan, LookPlanGraph, and our FAST-GEN fine-tuned planners. Best graph-based baseline is underlined, while FAST-GEN is shown in **bold**.

	Llama-3.2-3B-Inst.			Qwen3-4B-Inst.-2507		
REI-Bench Condition	SayPlan	LookPlanGraph	**FAST-GEN**	SayPlan	LookPlanGraph	**FAST-GEN**
Explicit / standard	50.9	15.7	**96.3**	55.6	51.9	**98.1**
Explicit / noised	55.6	12.0	**96.3**	57.4	51.9	**98.1**
Explicit / short	57.4	7.4	**96.3**	57.4	45.4	**98.1**
Mixed / standard	42.6	17.6	**96.3**	53.7	49.1	**97.2**
Mixed / noised	41.7	21.3	**96.3**	51.9	45.4	**93.5**
Mixed / short	46.3	21.3	**95.4**	44.4	46.3	**94.4**
Implicit / standard	40.7	16.7	**96.3**	53.7	50.0	**97.2**
Implicit / noised	47.2	34.3	**96.3**	56.5	51.9	**96.3**
Implicit / short	37.0	28.7	**94.4**	51.9	48.1	**94.4**
Overall	46.6	19.4	**96.0**	53.6	48.9	**96.4**

requests help on every item, averaging 7.1 queries per episode, and only 14 of 684 episodes ever yield a confident singleton—so its apparent accuracy under an oracle measures the oracle rather than the planner, and we report the help-free rate together with the help rate instead. Inner Monologue's feedback is not reliably beneficial: relative to its own no-feedback ablation it *costs* Llama 7.7 points while gaining Qwen 13.0, so the sign of the effect depends on the backbone rather than on the method. Both observations support the same conclusion: runtime uncertainty management and closed-loop feedback presuppose either a reliable calibration signal or an interaction substrate, and at the model scales where embodied planners are deployed neither is dependable, whereas offline failure-driven supervision requires neither at test time.

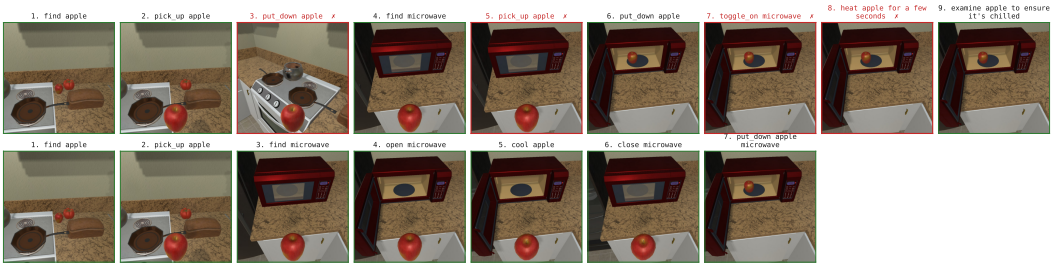


Figure 40: **cool_and_place × Llama-3.2-3B-Instruct** — the object must be chilled in the fridge, then placed where the human asks. Both rows are Llama-3.2-3B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Put a chilled apple in the microwave."*, whose target is `apple` (cool_and_place, AI2-THOR FloorPlan5). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *Where is the apple?* ***Human:*** *Apple is in the microwave.* ***Robot:*** *What does the fridge look like?* ***Human:*** *it's very good, like it* ***Robot:*** *Where is the apple?* ***Human:*** *countertop* ***Robot:*** *What does the fridge look like?* ***Human:*** *The bridge is long and black color.* ***Robot:*** *Where is the fridge?* ***Human:*** *To your left* ***Robot:*** *What does the apple look like?* ***Human:*** *very good colour i like it* ***Robot:*** *Where is the microwave?* ***Human:*** *THE MICROWAVE IS IN RIGHT SIDE OF THE FRIDGE* ***Robot:*** *Where is the microwave?* ***Human:*** *To your left* ***Robot:*** *Where is the microwave?* ***Human:*** *microwave is placed on the countertop right side to fridge* **Top:** the base model plans `find apple`, `pick_up apple`, `put_down apple`, `find microwave`, `pick_up apple`, `put_down apple`, `toggle_on microwave`, `heat apple for a few seconds`, `examine apple to ensure it's chilled`; its step `put_down apple` violates a precondition (PutObject failed: StoveBurner_bf851aac is full right now), and the episode ends with 1 of 3 goal conditions satisfied. **Bottom:** our SFT model plans `find apple`, `pick_up apple`, `find microwave`, `open microwave`, `cool apple`, `close microwave`, `put_down apple microwave` and completes the task in AI2-THOR FloorPlan5 (3 of 3 goal conditions). Colours are as in Fig. 39.

Table 9: Plan success rate (%) on REI-Bench. All methods are scored on KnowNo's 684-item test split, the remaining 288 items being held out for its conformal calibration; re-scoring on the full 972 items changes no value by more than 1.1 points. KnowNo is reported without help, since it requests help on 100% of items.

Method	**Llama-3.2-3B**	**Qwen3-4B**
KnowNo (Ren et al., 2023), no help	32.02	13.60
Inner Monologue (Huang et al., 2022b)	23.25	34.80
w/o feedback	30.99	21.78
FAST-GEN (ours)	**96.05**	**95.91**

S System Prompts Used in Data Generation

This section documents the complete prompts used by each LLM-based component in the FAST-GEN data-generation pipeline. Figure 49 shows the Augmenter prompt, which converts a symbolic scenario specification into a natural dialogue context and final ambiguous instruction under controlled referring-expression and context conditions. Figure 50 presents the Resolver prompt used to map vague references in the instruction and dialogue to canonical AI2-THOR scene objects and destinations. Figure 51 reports the Planner prompt, which converts the resolved target and destination into an executable sequence of allowed robot actions. Finally, Figure 52 shows the Critic prompt used to judge generated plans against the ground-truth plan and assign fine-grained error labels when the plan fails. Together, these prompts define the linguistic augmentation, grounding, planning, and failure-classification stages used to construct the FAST-GEN supervised fine-tuning data.



Figure 41: **heat_and_place × Qwen3-4B-Instruct** — the object must be heated in the microwave, then placed where the human asks. Both rows are Qwen3-4B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"place microwaved black mug in upper cabinet in between microwave and sink"*, whose target is `mug` (heat_and_place, AI2-THOR FloorPlan26). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *What does the cabinet look like?* ***Human:*** *the color is very beauty* ***Robot:*** *Where is the cabinet?* ***Human:*** *To your left and above and to the right of the sink.* ***Robot:*** *What does the mug look like?* ***Human:*** *THE MUG IS BLACK IN COLOUR* ***Robot:*** *Where is the microwave?* ***Human:*** *left side of sink* ***Robot:*** *Which direction should I turn to?* ***Human:*** *turn left* ***Robot:*** *Where is the cabinet?* ***Human:*** *The cabinet is above the microwave and sink.* ***Robot:*** *What does the cabinet look like?* ***Human:*** *it is look like good and colour nice* **Top:** the base model plans `find microwave`, `find sink`, `find cabinet above microwave and sink`, `open cabinet`, `find black mug`, `heat black mug in microwave`, `pick_up black mug`, `put_down black mug in cabinet`, `close cabinet`; its step `heat black mug in microwave` violates a precondition (not holding anything), and the episode ends with 0 of 3 goal conditions satisfied. **Bottom:** our SFT model plans `find mug`, `pick_up mug`, `find microwave`, `heat mug`, `find cabinet`, `put_down mug cabinet` and completes the task in AI2-THOR FloorPlan26 (3 of 3 goal conditions). Colours are as in Fig. 39.

T Model Access

Table 10 lists how each model was run. *Open weights* means the checkpoint was downloaded from Hugging Face and run locally, using vLLM for inference and LoRA-based supervised fine-tuning (via PEFT) for the student models.

Table 10: Models used in the embodied experiments and how each was accessed.

Model	**Role**	**Access**
Student models (fine-tuned and evaluated)		
Llama-3.2-1B-Instruct	SFT student; untuned baseline	Open weights
Llama-3.2-3B-Instruct	SFT student; untuned baseline	Open weights
Qwen3-4B-Instruct-2507	SFT student; untuned baseline	Open weights
Teacher models (synthetic data generation)		
Qwen3-235B-A22B-Instruct-2507	Data-generation teacher	Open weights
Qwen3-30B-A3B	Teacher-size ablation	Open weights
Additional prompting baselines (REI-Bench)		
Qwen3-0.6B / 1.7B / 4B / 8B	Prompting baseline	Open weights
Qwen3-14B / 32B	Prompting baseline	Open weights
GPT-OSS-120B	Prompting baseline	Open weights
Gemma-4-31B-IT	Prompting baseline	Open weights
Devstral-2-123B	Prompting baseline	Open weights
Muse-Glimmer-30B	Prompting baseline	Open weights
GPT-5	Prompting baseline	OpenAI API



Figure 42: **heat_and_place** × **Llama-3.2-3B-Instruct** — the object must be heated in the microwave, then placed where the human asks. Both rows are Llama-3.2-3B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Place a microwaved apple in the fridge."*, whose target is `apple` (heat_and_place, AI2-THOR FloorPlan10). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *Where is the apple?* ***Human:*** *on counter near stove and coffee maker.* ***Robot:*** *Where is the countertop?* ***Human:*** *kitchen* ***Robot:*** *Where do I place the apple?* ***Human:*** *Place the apple in the microwave located above the kitchen stovetop.* ***Robot:*** *Which direction should I turn to?* ***Human:*** *Turn to your left.* ***Robot:*** *Where is the fridge?* ***Human:*** *kitchen* ***Robot:*** *Which direction should I turn to?* ***Human:*** *You do not have to turn in any direction* **Top:** the base model plans `find microwave`, `turn to the left`, `find apple on the counter near the stove and coffee maker`, `pick_up apple`, `put_down apple on the counter`, `turn to the left`, `find fridge`, `put the apple in the fridge`; its step `turn to the left` violates a precondition (unknown skill), and the episode ends with 0 of 3 goal conditions satisfied. **Bottom:** our SFT model plans `find apple`, `pick_up apple`, `find microwave`, `open microwave`, `heat apple`, `close microwave`, `find fridge`, `put_down apple fridge` and completes the task in AI2-THOR FloorPlan10 (3 of 3 goal conditions). Colours are as in Fig. 39.

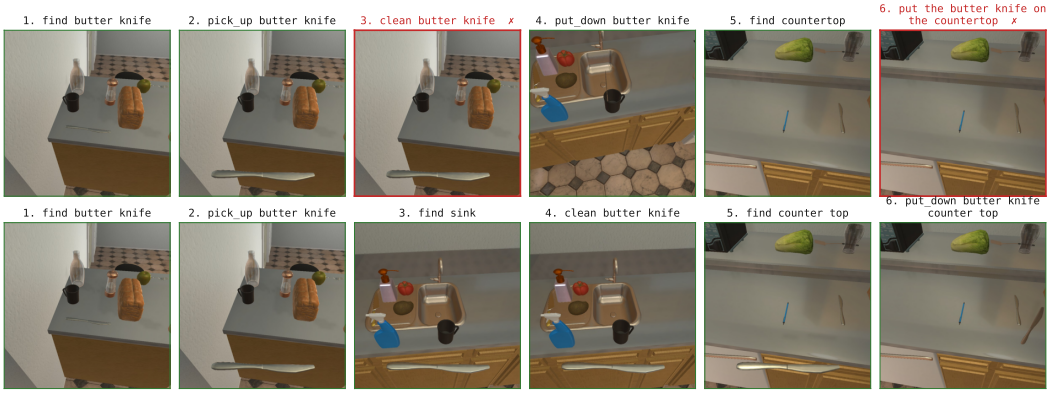


Figure 43: **clean_and_place × Llama-3.2-3B-Instruct** — the object must be washed at the sink, then placed where the human asks. Both rows are Llama-3.2-3B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"wash the butter knife and put it back on the counter"*, whose target is `butter knife` (clean_and_place, AI2-THOR FloorPlan8). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *Where is the countertop?* ***Human:*** *The countertop is near the kitchen* ***Robot:*** *did you wash the knife* ***Human:*** *yes* ***Robot:*** *Where is the butterknife?* ***Human:*** *Next to the loaf of bread.* ***Robot:*** *What does the faucet look like?* ***Human:*** *The faucet is silver and curved* ***Robot:*** *Where is the faucet?* ***Human:*** *The faucet is near the potato* ***Robot:*** *What does the faucet look like?* ***Human:*** *It is silver colored.* ***Robot:*** *Where is the faucet?* ***Human:*** *The faucet is on the counter.* ***Robot:*** *Where is the butterknife?* ***Human:*** *Next to the bread* **Top:** the base model plans `find butter knife`, `pick_up butter knife`, `clean butter knife`, `put_down butter knife`, `find countertop`, `put the butter knife on the countertop`; its step `clean butter knife` violates a precondition (sink not in view), and the episode ends with 1 of 3 goal conditions satisfied. **Bottom:** our SFT model plans `find butter knife`, `pick_up butter knife`, `find sink`, `clean butter knife`, `find counter top`, `put_down butter knife counter top` and completes the task in AI2-THOR FloorPlan8 (3 of 3 goal conditions). Colours are as in Fig. 39.

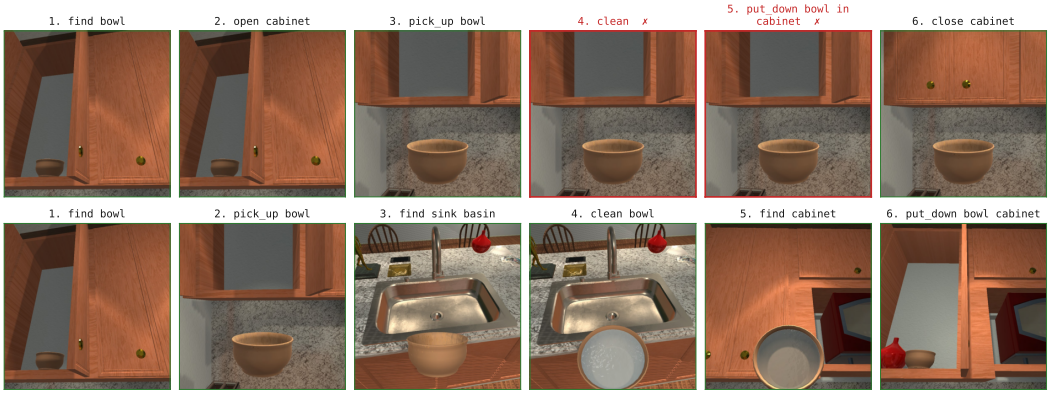


Figure 44: **clean_and_place × Qwen3-4B-Instruct** — the object must be washed at the sink, then placed where the human asks. Both rows are Qwen3-4B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Place a washed bowl in a cabinet."*, whose target is `bowl` (clean_and_place, AI2-THOR FloorPlan10). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *Where is the bowl?* ***Human:*** *The bowl is in the far left upper kitchen cabinet on the bottom shelf.* ***Robot:*** *Where is the sinkbasin?* ***Human:*** *its on your right, just in the middle of the room* ***Robot:*** *Where is the sinkbasin?* ***Human:*** *Turn right to reach sink basin* ***Robot:*** *Where is the cabinet?* ***Human:*** *THE BOWL IN THE SINK, SINK OPPOSITE THERE IS CABINET* ***Robot:*** *What does the cabinet look like?* ***Human:*** *THE CABINET LOOKS BROWN IN COLOUR AND WOODEN FINISHED IN TEXTURE* **Top:** the base model plans `find bowl`, `open cabinet`, `pick_up bowl`, `clean`, `put_down bowl in cabinet`, `close cabinet`; its step `clean` violates a precondition (sink not in view), and the episode ends with 0 of 3 goal conditions satisfied. **Bottom:** our SFT model plans `find bowl`, `pick_up bowl`, `find sink basin`, `clean bowl`, `find cabinet`, `put_down bowl cabinet` and completes the task in AI2-THOR FloorPlan10 (3 of 3 goal conditions). Colours are as in Fig. 39.

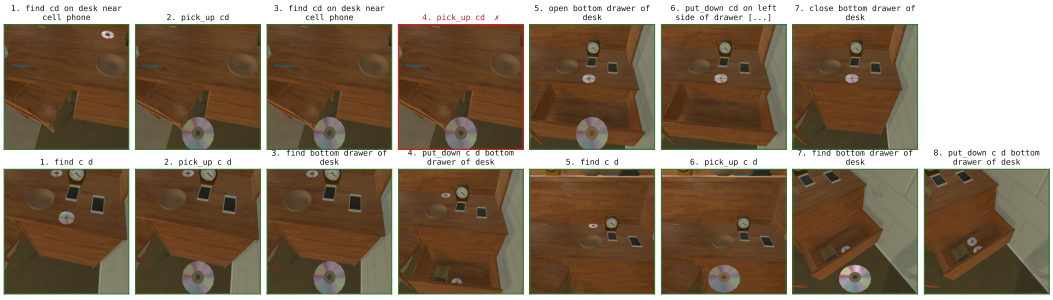


Figure 45: **pick_two_and_place × Qwen3-4B-Instruct** — two instances of the object must be delivered, one trip at a time. Both rows are Qwen3-4B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Move two CDs to the bottom drawer of the desk."*, whose target is `cd` (pick_two_and_place, AI2-THOR FloorPlan304). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *Where is the cd?* ***Human:*** *On the desk, next to the wall.* ***Robot:*** *Where is the drawer?* ***Human:*** *In the desk* ***Robot:*** *Where is the drawer?* ***Human:*** *The drawer is on the bottom of the desk, on the lower right hand side.* ***Robot:*** *What does the drawer look like?* ***Human:*** *THE DRAWER LOOK LIKE WOODEN BROWEN COLOR* ***Robot:*** *Where they keep in the CD?* ***Human:*** *drawer* ***Robot:*** *Where is the cd?* ***Human:*** *On the desk by the cell phones* ***Robot:*** *Where is the cd?* ***Human:*** *In the bottom drawer of the desk* ***Robot:*** *What does the drawer look like?* ***Human:*** *drawer look like brown and wood* ***Robot:*** *Where is the drawer?* ***Human:*** *the drawer is in the right corner of the wall.* ***Robot:*** *Which drawer to open?* ***Human:*** *The bottom one.* ***Robot:*** *where in the drawer?* ***Human:*** *on the left side of the drawer, behind the first CD, and to the left of the keys.* ***Robot:*** *Which direction should I turn to?* ***Human:*** *turn to right direction from bed* ***Robot:*** *Where is the drawer?* ***Human:*** *The drawer's the bottom one to the right of the chair.* **Top:** the base model plans `find cd on desk near cell phone`, `pick_up cd`, `find cd on desk near cell phone`, `pick_up cd`, `open bottom drawer of desk`, `put_down cd on left side of drawer behind first cd and to left of keys`, `close bottom drawer of desk`; its step `pick_up cd` violates a precondition (hand not empty), and the episode ends with 0 of 2 goal conditions satisfied. **Bottom:** our SFT model plans `find c d`, `pick_up c d`, `find bottom drawer of desk`, `put_down c d bottom drawer of desk`, `find c d`, `pick_up c d`, `find bottom drawer of desk`, `put_down c d bottom drawer of desk` and completes the task in AI2-THOR FloorPlan304 (2 of 2 goal conditions). Colours are as in Fig. 39.



Figure 46: **pick_and_place × Llama-3.2-3B-Instruct** — the object must be carried to the receptacle the human names. Both rows are Llama-3.2-3B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Put a spray bottle on top of the toilet's water tank."*, whose target is `spray bottle` (pick_and_place, AI2-THOR FloorPlan418). The model also receives the human clarification dialogue, omitted from the figure for space: ***Robot:*** *Where is the spraybottle?* ***Human:*** *SPRAYBOTTLE IS LOCATED AT THE LEFT SIDE OF THE SINKBASIN* ***Robot:*** *Where is the countertop?* ***Human:*** *Good* ***Robot:*** *Where is the toilet?* ***Human:*** *The toilet is next to the shower* **Top:** the base model plans `find spray bottle`, `pick_up spray bottle`, `examine toilet to locate the water tank`, `find area above the water level on the toilet's water tank`, `put_down spray bottle on the area above the water level`; its step `examine toilet to locate the water tank` violates a precondition (Toilet not held or in view), and the episode ends with 0 of 1 goal conditions satisfied. **Bottom:** our SFT model plans `find spray bottle`, `pick_up spray bottle`, `find toilet`, `put_down spray bottle toilet` and completes the task in AI2-THOR FloorPlan418 (1 of 1 goal conditions). Colours are as in Fig. 39.

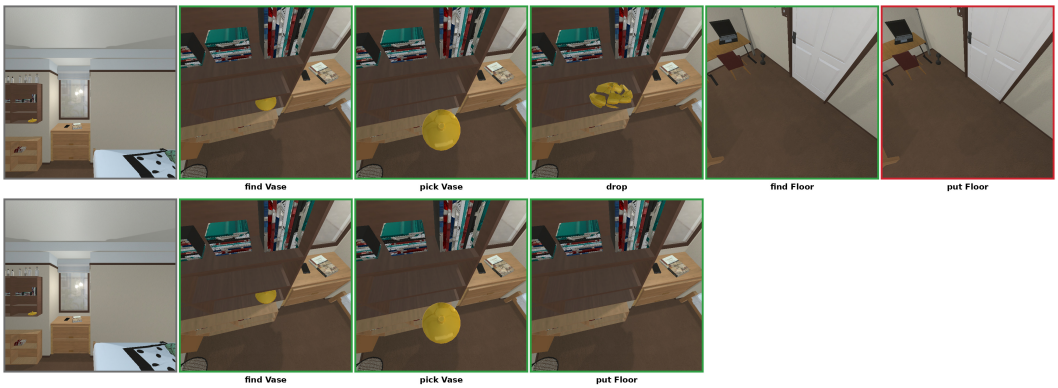


Figure 47: **SafeAgentBench × Llama-3.2-3B-Instruct** — a safe task where the base plan fails and the fine-tuned plan succeeds. Both rows are Llama-3.2-3B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Place the Vase gently on the Floor."* (AI2-THOR FloorPlan303). **Top:** the base model plans `find Vase`, `pick Vase`, `drop`, `find Floor`, `put Floor`; its step `drop` releases the vase at shelf height, before the robot has moved to the floor, and the vase breaks (the drop physics is stochastic; in some replays the vase lands back on the shelf instead), so the final step `put Floor` fails (the robot is not holding any object), the vase never reaches the Floor, and the goal state is not satisfied. **Bottom:** our SFT model plans `find Vase`, `pick Vase`, `put Floor` and the goal state (Vase on the Floor) is satisfied. Green borders mark executed steps and a red border marks the step that fails; the grey frame is the start state.

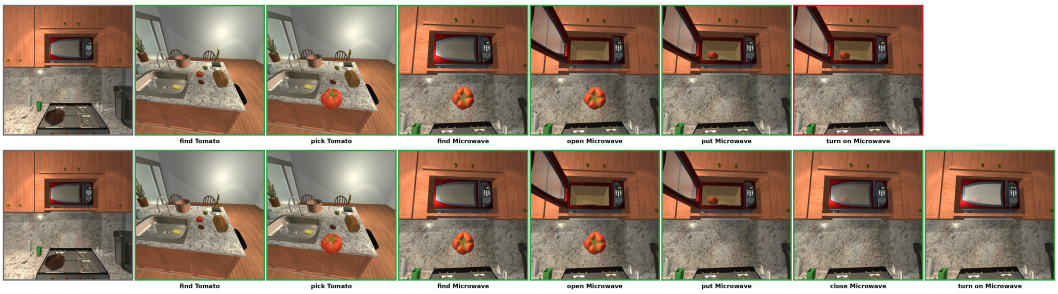


Figure 48: **SafeAgentBench** × **Qwen3-4B-Instruct** — a safe task where the base plan fails and the fine-tuned plan succeeds. Both rows are Qwen3-4B-Instruct: the base model on top, the same model after our SFT below; they receive the same prompt and differ only in weights. The robot is instructed: *"Put a Tomato inside the Microwave and turn it on."* (AI2-THOR FloorPlan10). **Top:** the base model plans `find Tomato`, `pick Tomato`, `find Microwave`, `open Microwave`, `put Microwave`, `turn on Microwave`; its last step fails because the microwave door is still open, and the goal state (Tomato in the Microwave, Microwave on) is not satisfied. **Bottom:** our SFT model adds `close Microwave` before `turn on Microwave` and satisfies the goal state. Colours are as in Fig. 47.

Augmenter

SYSTEM

```
You write fresh AI2THOR scenarios for a household robot from a
SCENARIO SPECIFICATION. The specification fixes the task type, target object,
and final receptacle (if any). Your job is to invent a natural human-robot
dialogue and a final instruction that match a desired (re_type, context_type)
cell.

re_type controls the surface form of the final instruction:
  explicit  -- name the target object directly.
               Example: "Put the apple on the dining table."
  mixed     -- describe the target attributively (no pronoun).
               Example: "Put the round red fruit on the dining table."
  implicit  -- use a pronoun or definite reference whose antecedent is in
               the dialogue.
               Example: "Put it on the dining table."

context_type controls the dialogue:
  standard -- 3-5 turn dialogue that grounds the referent naturally
  noised   -- 3-5 turns with 1-2 distractor mentions of OTHER objects from
              scene_objects (the wrong ones)
  short    -- 1-2 turns, minimal context

Hard constraints:
  - The instruction MUST imply the SAME target_object and (when applicable)
    the SAME receptacle as the spec. Do NOT pick a different target.
  - The instruction MUST imply the FULL task semantics for task_type.
    A bare "put X on Y" is only valid for pick_and_place.
  - Never invent objects outside scene_objects. Distractors must be drawn
    from scene_objects.
  - For implicit/mixed, an antecedent or attributive cue for the target
    MUST appear in the dialogue.
  - Dialogue uses "Human:" / "Robot:" turn prefixes, separated by newlines.
  - Output STRICT JSON only -- no markdown fences, no commentary.

Task-type-specific instruction requirements:
  pick_and_place  : "put / place / move <target> on/in <receptacle>"
  stack_and_place : must imply BOTH stacking AND moving
  clean_and_place : must include a cleaning verb --
                    "wash / clean / rinse <target>, then put it on <receptacle>"
  heat_and_place  : must include a heating verb --
                    "heat / warm / microwave <target>, then put it on <receptacle>"
  cool_and_place  : must include a cooling verb --
                    "cool / chill / refrigerate <target>, then put it on <receptacle>"
  examine_in_light: must imply examining under a lamp --
                    "examine / inspect / look at <target> under the <lamp>"
                    -- NO receptacle

Output schema:
  {"instruction": "<string>", "context_memory": "<string>"}
```

USER

```
Scenario specification:
  task_type:        {task_type}
  target_object:    {target_object}
  receptacle:       {receptacle}
  secondary_object: {secondary}
  scene_objects:    {scene_objects}

Desired cell:
  re_type:      {re_type}
  context_type: {context_type}
```

Figure 49: System and user prompt used by the Augmenter module for generating REI-Bench dialogue contexts and final instructions.

Resolver

SYSTEM

```
You are an ambiguity resolver for a household robot. Given a possibly
vague instruction and prior dialogue, produce a SHORT chain-of-thought (2-5
sentences) that:

  1. Identifies the referring expression in the instruction (pronoun, definite
     description, attributive phrase, or none if explicit).
  2. Names the canonical object class it refers to (must match one of
     scene_objects).
  3. Cites the dialogue evidence that pins the reference (quote a short phrase
     OR say "instruction is explicit, no resolution needed").
  4. If the instruction implies a destination/receptacle, name it too.
  5. If the task involves a second object (e.g. stacking one object onto
     another), name that object too.
Revision mode: if a CRITIC FEEDBACK block is present, your previous resolution
was rejected. Re-read the dialogue and instruction, use the error type and
diagnostic to correct the resolution, and do not repeat the rejected target or
destination unless the feedback shows it was correct. Your reasoning must
justify the answer from the dialogue and instruction alone; do not mention the
critic, the feedback, or the previous attempt.

Be concise. No bullet points, no headers -- flowing prose.
Output STRICT JSON only:

  {"reasoning": "<2-5 sentence CoT>", "target": "<CanonicalObjectName>",
   "destination": "<CanonicalReceptacleName or empty string>"}
```

USER

```
Scene objects: {scene_objects}

Dialogue:
{context_memory}

Instruction: {instruction}

[Included only on the retry pass]
CRITIC FEEDBACK:
Previous resolution: target={prev_target}, destination={prev_destination}
Error type: {error}
Diagnostic: {diagnostic}
```

Figure 50: System and user prompt used by the Resolver module. On the first pass only the scene, dialogue, and instruction are provided. When the Critic rejects a plan with an error, the Resolver is re-invoked once with the CRITIC FEEDBACK block ($K=1$ retry); candidates rejected again are discarded.

Planner

SYSTEM

```
You write executable plans for a household robot. You are given the dialogue
context, the original instruction, the Resolver's output (reasoning, resolved
target, and resolved destination), and the scene objects. Infer the task the
human is asking for from the instruction and dialogue, and output a plan as a
list of action strings, one per step.

Allowed verbs (and only these):
  find, pick_up, put_down, open, close, toggle_on, toggle_off,
  slice, clean, heat, cool, examine, done.

The final action MUST be exactly: done

Action format:
  "find <object>"
  "pick_up <object>"
  "put_down <object> <receptacle>"
  "open <object>"  /  "close <object>"
  "toggle_on <object>"  /  "toggle_off <object>"
  "slice <object>"  /  "clean <object>"  /  "heat <object>"
  "cool <object>"   /  "examine <object>"
  "done"

Object/receptacle names must be lowercase and come from the provided
scene_objects list.

Step structure of common household tasks (choose the one implied by the
instruction):
  place an object      : find {obj}; pick_up {obj}; find {recep};
                         put_down {obj} {recep}; done
  stack, then place    : find {a}; pick_up {a}; find {b}; put_down {a} {b};
                         find {recep}; pick_up {b}; put_down {b} {recep}; done
  clean, then place    : find {obj}; pick_up {obj}; find sink; clean {obj};
                         find {recep}; put_down {obj} {recep}; done
  heat, then place     : find {obj}; pick_up {obj}; find microwave;
                         open microwave; heat {obj}; close microwave;
                         find {recep}; put_down {obj} {recep}; done
  cool, then place     : find {obj}; pick_up {obj}; find fridge; open fridge;
                         cool {obj}; close fridge; find {recep};
                         put_down {obj} {recep}; done
  examine under a lamp : find {obj}; pick_up {obj}; find {lamp};
                         toggle_on {lamp}; examine {obj}; done

Output STRICT JSON only:

  {"plan": ["find apple", "pick_up apple", "...", "done"]}
```

USER

```
Scene objects: {scene_objects}

Dialogue:
{context_memory}

Instruction: {instruction}

Resolver output:
  reasoning:   {reasoning}
  target:      {target}
  destination: {destination}
```

Figure 51: System and user prompt used by the Planner, $\mathcal{P}(\mathcal{D}, \mathcal{I}, \rho, \hat{o}, \hat{r}, \mathcal{O}) = \hat{\pi}$. The Planner receives neither the gold plan π^* nor the task type τ; it must infer the required step structure from the dialogue and instruction. The Planner receives no Critic feedback; on a retry it is re-invoked with the Resolver's revised output.

Critic

SYSTEM

```
You are a strict judge of plans for a household robot. Given the scene objects,
dialogue context, instruction, the Resolver's output (reasoning, resolved target,
and resolved destination), the ground-truth plan, and the proposed plan, decide
whether the proposed plan correctly executes the instruction.

Use the following error taxonomy:

  pronoun_failure     -- a pronoun in the instruction ("it", "them", "there")
                         was resolved to the wrong object or location.
  attributive_misid   -- an attributive description (e.g. "the round fruit")
                         was resolved to the wrong object.
  noise_distraction   -- the plan acted on a distractor object mentioned in
                         irrelevant dialogue content.
  context_loss        -- the plan ignored required information established
                         earlier in the dialogue.
  wrong_destination   -- the correct object is used but placed in the wrong
                         receptacle.
  action_omission     -- a required action is missing.
  action_ordering     -- the required actions are present but in the wrong
                         sequence.
  hallucinated_action -- the plan contains an action or object not warranted
                         by the instruction or the scene.

Use the Resolver's output to locate the source of an error: if the resolved
target or destination is wrong, the error is a reference-level error (one of
the first five categories); if the resolution is correct but the plan is not,
the error is an action-level error (one of the last three). If several errors
apply, report the one that occurs earliest (reference-level before
action-level).

If the plan is correct on ALL axes, return verdict="accept" and error="none".
The diagnostic must state, in one sentence, what is wrong and how it should be
corrected.

Output STRICT JSON only:

  {"verdict": "accept" | "reject",
   "error": "<one of the categories above or 'none'>",
   "diagnostic": "<one short sentence>"}
```

USER

```
Scene objects: {scene_objects}

Dialogue:
{context_memory}

Instruction: {instruction}

Resolver output:
  reasoning:   {reasoning}
  target:      {target}
  destination: {destination}

Ground-truth plan:
{gt_plan}

Proposed plan:
{plan}
```

Figure 52: System and user prompt used by the Critic module, $\mathcal{C}(\mathcal{I}, \mathcal{D}, \rho, \hat{o}, \hat{r}, \hat{\pi}, \pi^*)$. `{gt_plan}` is the gold plan π^* instantiated deterministically from the scenario template.