Rewrite Before You Call: Single-Pass Context Resolution for On-Device Multi-Turn Mobile API Calling

Anonymous ACL submission

Abstract

Modern mobile assistants need to resolve implicit meaning in multi-turn dialogue, such as deixis, ellipsis, and long-range coreference, to execute API actions. However, on-device small language models often struggle with context-dependent follow-up requests under deployment constraints. We propose Rewrite-Enhanced Autoregressive Decomposition (READ), a single-pass framework that uses context-resolved rewriting as explicit supervision and generates the rewrite before the API name and arguments within the same autoregressive sequence. We introduce a 17k controlled multi-turn mobile API dialogue benchmark with paired rewrites and turn-level API annotations, together with a separate 1k human-written realistic validation set for evaluation under noisier user-style utterances. Across 4B-class and smaller models, READ consistently improves API-call exact-match accuracy over history-based baselines and lightweight rewrite-then-plan alternatives, with larger gains on longer-range contextual dependencies. Additional evaluations show that READ remains competitive on external function-calling benchmarks while offering a favorable one-pass accuracy–latency trade-off for on-device deployment.

1 Introduction

Voice-based AI assistants such as Samsung Bixby (Samsung Electronics, 2025) or Apple Siri (Apple Inc., 2024) have become essential tools in modern mobile environments, enabling users to interact with a variety of services such as messaging, scheduling, and navigation through natural language. To execute requests, the assistants must accurately understand user utterances and reliably map them to appropriate internal Application Programming Interface (API) calls with the correct parameters. Although large language models (LLMs) have made significant progress in planning and decision-making tasks, most tool-use evaluations and deployments still emphasize single-turn or short-context settings (Shen et al., 2025; Xie et al., 2024), leaving the harder regime of realistic multi-turn mobile dialogues underexplored. In practice, users frequently issue follow-up requests that rely on prior turns and are underspecified (e.g., deictic pronouns and omitted arguments), which is particularly challenging for on-device small language models (SLMs) under strict compute and memory budgets (Duan et al., 2024).

Prior benchmarks and tool-augmented dialogue frameworks (Huang et al., 2024; Patil et al., 2024; Qin et al., 2024; Li et al., 2023) have made strides toward evaluating API-based planning. However, they primarily focus on single-turn inputs, simplified workflows, or server-scale models, making them poorly suited for the dynamic, incremental nature of mobile interactions. As a result, they do not directly stress-test the ambiguity, long-range coreference, and intent drift that arise in real mobile follow-ups—failure modes that are especially costly for on-device assistants that must operate autonomously, with minimal external computation (Parikh et al., 2019; Zhou et al., 2019).

To address these limitations, we introduce a multi-turn mobile API dialogue benchmark with paired context-resolved rewrites and turn-level API annotations. Using this benchmark, we propose Rewrite-Enhanced Autoregressive Decomposition (READ), a single-pass framework that first generates a rewritten query as an explicit intermediate output and then predicts the API name and arguments within the same autoregressive sequence. This design makes context resolution explicit before API prediction, improving robustness in context-dependent mobile API dialogue.

Figure 1(a) illustrates the key differences among single-turn methods with multiple requests, conventional multi-turn methods, and our proposed rewriting-based method. While single-turn execu-

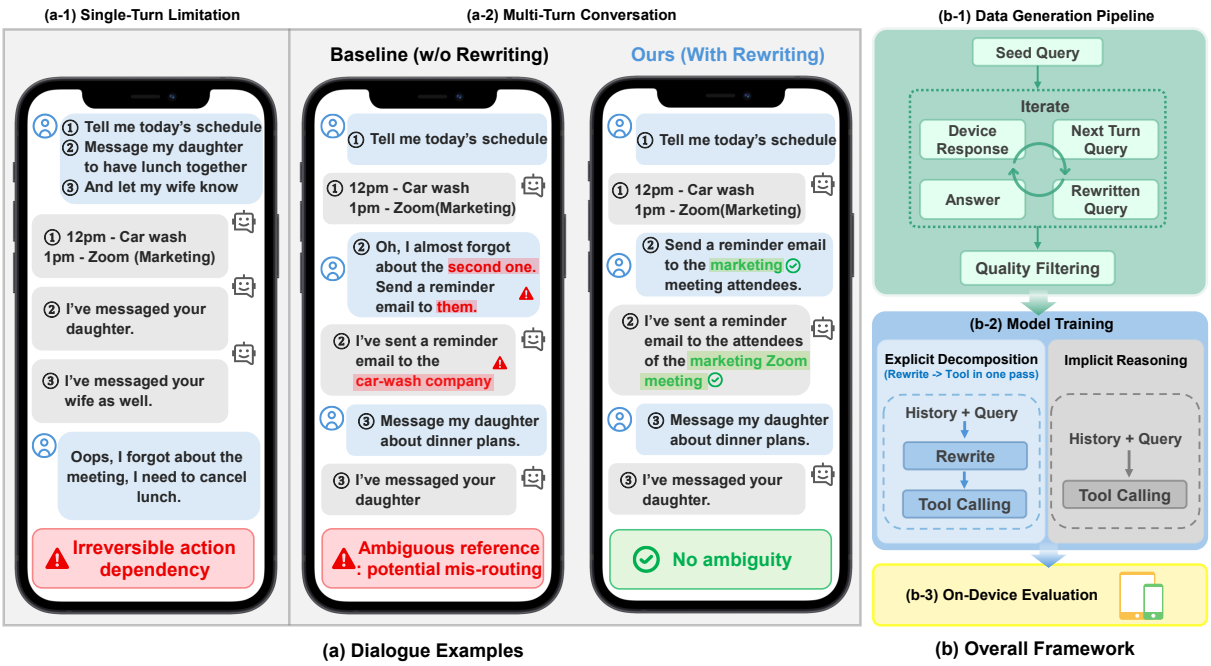


Figure 1: (a) Comparison of single-turn and multi-turn interactions, highlighting ambiguity in multi-turn settings and the role of rewriting. (b) Overall framework including data generation, model training, and on-device evaluation.

tion can produce unintended outcomes for bundled requests and history-based multi-turn methods may still suffer from ambiguity, our approach explicitly rewrites each turn to clarify the current intent before API prediction.

Our main contributions are as follows. (1) We propose READ, a single-pass explicit decomposition framework that places context-resolved rewriting and API prediction in one autoregressive output sequence for on-device mobile API calling. (2) We construct a 17k controlled multi-turn mobile API dialogue benchmark with paired rewrites and turn-level API annotations, and further provide a separate 1k human-written realistic validation set to test robustness under noisier user-style follow-up requests. (3) We show that READ improves API-call exact-match accuracy over history-conditioned baselines and lightweight rewrite-then-plan alternatives, with gains increasing as dialogue depth and reference distance grow. (4) We analyze the one-pass accuracy–latency trade-off on mobile hardware and release an open-source evaluator for reproducible deployment-oriented assessment, with code, model, and on-device evaluation resources provided in Appendix A.

2 Related Work

Recent tool-use and function-calling benchmarks such as MetaTool, APIBench, ToolBench, APIBank, and BFCL v3/v4 have provided important foundations for evaluating tool selection, API planning, and fine-grained function-call correctness (Huang et al., 2024; Patil et al., 2024; Qin et al., 2024; Li et al., 2023; Patil et al., 2025). Mobile-oriented benchmarks such as ShortCuts-Bench, DroidCall, and HammerBench further address mobile API execution and realistic function-calling scenarios (Shen et al., 2025; Xie et al., 2024; Wang et al., 2025). However, these benchmarks collectively focus on general tool-use ability, single-turn or short-context execution, clarification-oriented evaluation, or multi-step tool-use workflows. In contrast, our work targets context-dependent mobile follow-up utterances, with paired context-resolved rewrites directly tied to API-call exact match.

Deploying language models in on-device environments involves strict latency and memory constraints, making it impractical to directly use large models on mobile devices (Yan and Ding, 2025; Li et al., 2024; Husom et al., 2025; Xiao et al., 2025). However, enabling small models to reliably handle such dependencies under these constraints remains underexplored. Our work addresses this gap by studying whether explicit context resolution can improve mobile API planning with 4B-class and smaller models under these constraints.

Query rewriting has been widely studied for in-

formation retrieval (Mo et al., 2023; Kostric and Balog, 2024; Ma et al., 2023; Ye et al., 2023), multi-step reasoning (Shi et al., 2024; Shao et al., 2023), and assistant-oriented dialogue (Lu et al., 2023). In particular, 5IDER addresses voice-assistant query reformulation involving intent carryover, disfluencies, entity carryover, and repair. However, applying such rewriting to mobile API prediction as a conventional rewrite-then-plan pipeline would introduce an additional rewriting stage before planning, which is costly under on-device SLM constraints. READ instead treats rewriting as part of the API-call prediction output, avoiding a separate rewriting module.

3 Dataset and Methodology for READ

3.1 Mobile Dialogue Multi-Turn Dataset

For controlled training and evaluation, we construct a new Mobile Dialogue Multi-Turn dataset tailored to tool-augmented conversations on mobile devices, as shown in Figure 1(b). Figure 1(b-1) illustrates the pipeline used for generating and filtering this dataset. This dataset comprises 17,453 turn-level multi-turn dialogue instances that simulate realistic mobile-assistant interactions involving API calls. We use this dataset as a controlled testbed for context-dependent API prediction, rather than as a replacement for real production mobile-assistant logs.

We generate and annotate each dialogue via a four-step iterative procedure, beginning with an initial user query: **(1) Assistant response generation**: Given the initial user query, the assistant response is generated by verbalizing the simulated outcome of the corresponding API execution. **(2) Next user query generation**: The user's next query is generated from the assistant's response, simulating realistic follow-up interactions. **(3) Query rewriting**: The user's query is explicitly rewritten by integrating relevant information from previous turns, clearly expressing the user's intent in a single, context-aware sentence. **(4) API and parameters annotation**: Each rewritten query is annotated with the corresponding API call and parameters needed for execution.

We select 41 APIs relevant to mobile assistant scenarios involving multi-turn interactions by reviewing the 26 APIs defined in Android's official Intent documentation (Google, 2025) and the DroidCall dataset. To comprehensively cover realistic usage contexts, we extend this initial selection with additional APIs addressing common mobile tasks, such as messaging, scheduling, and media management. Each dialogue begins with an initial user query based on the mobile API invocation setting of DroidCall (Xie et al., 2024) and then evolves through 1–6 user–assistant turns with interleaved utterances and API actions. Further details on the full API set, API descriptor format, and prompt templates for the dataset generation pipeline are provided in Appendix B.

Generator	Dialogue Quality Filter ↓	Annotation Quality Filter ↓	Human Pass Rate ↑
Gemini-2.0	29.38	3.50	90.2
GPT-4.1	37.15	10.38	90.9
GPT-5-mini	36.69	2.04	92.5
o4-mini	16.93	2.28	95.1

Table 1: Quality assessment of candidate data generators. Dialogue and Annotation Quality Filters report average filtering rates for contextual/API/rewrite quality and linguistic/API-parameter correctness, respectively. Lower filtering rates and higher human pass rates indicate better quality.

3.2 Multi-Stage Quality Filtering

To mitigate artifacts and annotation errors in LLM-generated synthetic dialogues, we apply a multi-stage quality filtering pipeline that combines generator comparison, automated validation, generator-selection human assessment, and final dataset human validation.

As the first stage, we conducted comparative experiments to identify a suitable generator for multi-turn mobile API dialogues. We compared multiple candidate generators and selected the best-performing model using a hybrid quality assurance protocol. Building on prior analyses of LLM-generated data quality (Stacey et al., 2024; Xu et al., 2023a), we adopted a hybrid evaluation that combined human assessments and LLM-based evaluations to compare the following models: o4-mini (OpenAI, 2025c), GPT-5-mini (OpenAI, 2025b), GPT-4.1 (OpenAI, 2025a), and Gemini-2.0-Flash (Google DeepMind, 2025). Evaluation criteria included: (1) proper reference to previous dialogue, (2) alignment of queries with target APIs, (3) accurate rewriting of queries for their intended purpose, (4) linguistic consistency, and (5) correct API parameter annotation.

Initially, we filtered the generated data through automated LLM-based evaluations according to the stated criteria. As shown in Table 1, o4-mini

showed the strongest overall profile for dialogue generation, achieving the lowest dialogue-quality filtering rate and the highest human pass rate. Although GPT-5-mini had a slightly lower annotation-quality filtering rate, o4-mini provided the best balance between contextual dialogue quality and human-validated reliability. We therefore selected o4-mini to generate the final dataset, followed by the additional validation described below.

After constructing the final dataset, we conducted a separate human validation study to assess whether the generated instances are usable for evaluating context-dependent API prediction. Specifically, we evaluated 2,000 stratified samples from the final dataset. Ten trained evaluators judged each instance using three binary criteria: API alignment, reference resolution, and argument groundability. As detailed in Appendix C, 93.8% of the sampled instances passed all three criteria.

We emphasize that this validation checks the correctness of controlled benchmark annotations, while the separate realistic validation set evaluates robustness to noisier user-style utterances. Additional details on the filtering criteria, human evaluation protocol, evaluator composition, agreement analysis, and invalid-case error distribution are provided in Appendices C.1–C.3.

3.3 Single-Pass Explicit Decomposition for Context Clarity

Conventional multi-turn API planning approaches directly map the dialogue history and the current user utterance to the target API and its arguments. However, this direct formulation entangles context resolution with API planning, making the prediction process particularly challenging for small language models.

To address this issue, we propose a *single-pass explicit decomposition* approach. The key idea is to make context resolution explicit before API planning, while keeping both steps within a single autoregressive output sequence.

The dialogue history is defined as:

$$C_t = (U_1, S_1, U_2, S_2, \ldots, U_{t-1}, S_{t-1}),$$

where U_t denotes the current user utterance. A conventional approach directly performs planning based on the dialogue history and the current utterance:

$$(API_t, Param_t) = f_\theta(C_t, U_t).$$

In contrast, our method generates a single output sequence Y_t:

$$\begin{aligned} Y_t &= g_\theta(C_t, U_t) \\ &= [\text{Rewrite: } \hat{U}_t;\ \text{Plan: } API_t;\ \text{Args: } Param_t]. \end{aligned}$$

Here, $\hat{U}_t$ denotes the rewritten query, which resolves the current utterance U_t using the dialogue history C_t. This output sequence separates context resolution from API planning while preserving a single autoregressive generation process.

This formulation naturally aligns with autoregressive generation. Since the model generates the output from left to right, the prediction can be factorized as:

$$\begin{aligned} P(Y_t \mid C_t, U_t) = P(\hat{U}_t \mid C_t, U_t) \\ \cdot P(API_t, Param_t \mid C_t, U_t, \hat{U}_t). \end{aligned}$$

The generated rewritten query $\hat{U}_t$ serves as a strong prefix condition for the subsequent planning step. Therefore, planning does not rely solely on implicit reasoning over the dialogue history; instead, it can be conditioned on an explicit query that the model has already generated.

The following example illustrates this sequential conditioning process. Suppose that John's phone number, +1-202-555-0147, has been mentioned in the previous dialogue history and the user then says, "Call that person." The model generates a single output sequence:

```
Rewrite: Call +1-202-555-0147
Plan: dial
Args: phone_number = "+1-202-555-0147"
```

We hypothesize that this ordering preserves the benefits of explicit rewriting even when rewriting and planning are generated in a single pass.

3.4 Training Setup for Small Language Models

Deploying large-scale models directly on mobile devices remains challenging due to latency and memory constraints (Yan and Ding, 2025; Li et al., 2024; Husom et al., 2025; Xiao et al., 2025). Therefore, we select six relatively small open-source LLMs suitable for on-device deployment: Qwen3, Phi-4, Llama-3.2, SmolLM3, Granite3.3, and GLM-edge (Yang et al., 2025; Microsoft et al., 2025; Meta, 2024; Hugging Face, 2025; IBM, 2025; THUDM, 2024).

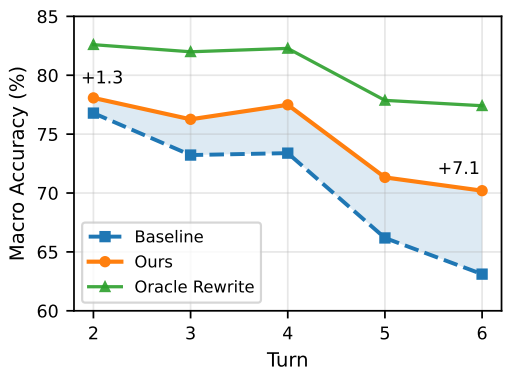


Figure 2: Average macro accuracy across turns 2–6 over six models. Ours consistently outperforms the Baseline, while Oracle Rewrite indicates the remaining headroom from perfect context resolution.

For all models, we perform LoRA-based supervised fine-tuning (Hu et al., 2022; Han et al., 2024; Xu et al., 2023b) on our multi-turn dialogue dataset to reduce training overhead while adapting the models to multi-turn API dialogue.

4 Experiments

4.1 Experimental Setup

4.1.1 Compared Methods

To isolate the effect of explicit decomposition, we compare history-based baselines with rewrite-based methods.

Oracle Rewrite directly feeds the gold rewritten query into the planner, serving as an upper bound when rewrite quality is assumed to be perfect.

Schema-aware Planner (Baseline) directly predicts the API name and arguments from the dialogue history, current query, and API schema, representing a standard history-conditioned function-calling baseline following prior tool-use and function-calling settings (Li et al., 2023; Patil et al., 2025).

Schema-aware Planner + Resampling applies schema-level correction by resampling invalid outputs up to three times with temperatures of 0, 0.2, and 0.5. This variant is related to prior work on output-side constrained generation for faithful API-call prediction (Wang et al., 2024).

Schema-aware Planner + Retrieval replaces the full dialogue history with the top-2 retrieved turns most relevant to the current query. This retrieval variant tests whether retrieval-based history selection can substitute for explicit rewriting (Lewis et al., 2020; Mo et al., 2023; Ma et al., 2023).

Two-stage QR is a rewrite-then-plan comparison method inspired by 5IDER-style query rewriting. It uses a separately trained rewrite-only model, while keeping the planner identical to the Oracle Rewrite setting (Lu et al., 2023).

READ (Ours) follows the single-pass explicit decomposition defined in Section 3.3.

READ + Resampling further applies the same schema-level resampling to our method.

4.1.2 Evaluation Protocol

All train, validation, and test partitions are constructed at the dialogue level to prevent turns from the same dialogue from appearing across different splits.

We report macro API-call exact-match accuracy over Turns 2–6. A prediction is counted as correct only when the API name and all required arguments exactly match the gold annotation; accuracies are computed per API-call category and then macro-averaged.

Additional details are provided in Appendix D.

4.2 Experimental Results

Overall Performance. As shown in Table 2, the proposed READ method (Ours) consistently improves performance over the Baseline. In particular, Ours achieves higher average accuracy than comparison methods with additional techniques, such as constraint-guided resampling and retrieval-augmented history, indicating that the main source of improvement comes from explicit context resolution rather than simple output correction or history compression. A paired non-parametric bootstrap over test examples further confirms the improvement, with a 95% CI of [2.6, 4.2] points and positive point estimates for all six models.

Figure 2 further shows that turn-wise accuracy tends to decline as dialogue turns accumulate, suggesting that context resolution becomes more difficult in later turns. Even under these challenging conditions, Ours consistently maintains higher performance than the Baseline across all turns. Finally, Oracle Rewrite, which directly feeds the gold rewritten query into the planner, provides an upper-bound performance that could be achieved with improved rewriting quality.

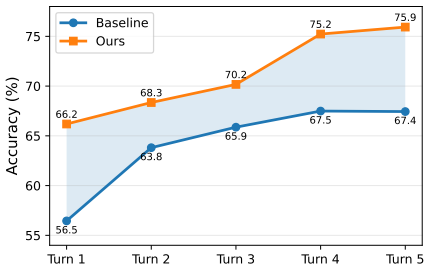
Comparison with Resampling and Retrieval Variants. Resampling yields only marginal gains (+0.5 for Baseline, +0.1 for READ) while adding decoding latency, and retrieval underperforms the

Method	Qwen3 (4B)	Phi-4 (4B)	Llama-3.2 (3B)	SmolLM3 (3B)	Granite3.3 (2B)	GLM-edge (1.5B)	Avg
Oracle Rewrite	82.7	76.4	81.2	80.3	81.7	80.3	80.4
Schema-aware Planner (Baseline)	73.6	67.0	72.2	67.2	73.9	69.3	70.5
Schema-aware Planner + Resampling	74.0	67.5	72.3	69.0	74.0	69.4	71.0
Schema-aware Planner + Retrieval	67.8	62.0	66.3	63.2	68.2	64.8	65.4
Two-stage QR	73.3	68.4	71.3	70.4	72.2	70.7	71.1
READ (Ours)	77.5	71.0	74.8	74.9	75.7	74.3	74.7
READ + Resampling	77.5	71.0	74.8	75.1	76.0	74.4	74.8

Table 2: Main API-call exact-match accuracy (%) across six target models.

Model	T2 Δ	T3 Δ	T4 Δ	T5 Δ	T6 Δ	Overall Δ
GLM-edge	+1.05	+4.25	+5.36	+4.65	+9.72	+5.00
Granite	-1.77	+2.00	+1.23	+2.78	+4.65	+1.77
Llama-3.2	-0.29	+2.26	+2.54	+3.90	+4.12	+2.51
Phi-4	+4.72	+1.93	+2.75	+4.99	+5.63	+4.00
Qwen3	+1.88	+1.65	+4.66	+6.36	+4.74	+3.86
SmolLM3	+2.16	+6.12	+8.08	+8.17	+13.65	+7.63
Avg.	+1.29	+3.04	+4.10	+5.14	+7.09	+4.13

(a) Turn-wise performance gap.



(b) Turn 6 accuracy by referenced turn.

Figure 3: Coreference analysis across dialogue depth and reference distance. In (a), Δ denotes Ours minus Baseline. In (b), earlier referenced turns indicate longer-range references.

Method	Llama-3.2 (3B)	SmolLM3 (3B)	GLM-edge (1.5B)	Avg
Baseline	68.3	62.9	68.1	66.4
Ours	74.3	70.6	72.7	72.5
Oracle Rewrite	82.9	83.7	83.7	83.4

Table 3: Accuracy (%) on the Realistic Validation Set.

full-history Baseline by omitting slot-bearing information distributed across the dialogue history.

Comparison with Two-stage QR. Two-stage QR shows that a separate rewrite-then-plan pipeline inspired by prior 5IDER-style query rewriting can improve over the basic Baseline. In the main comparison, we report Two-stage QR with a 0.6B rewrite-only model as a lightweight on-device-oriented two-call baseline. It improves over the Baseline but remains below READ, suggesting that simply adding a small separate rewriter is not sufficient to match single-pass joint decomposition. We further evaluate a stronger 4B two-stage rewriter in the accuracy–latency analysis. This stronger two-stage system slightly improves accuracy over READ, but requires an additional model call and increases end-to-end latency by 29.1%. Thus, our claim is not that single-pass READ always maximizes accuracy, but that it provides a favorable Pareto point for on-device deployment by capturing most of the benefit of explicit rewriting within one autoregressive pass.

Coreference Analysis. Figure 3 analyzes whether explicit decomposition provides larger benefits in longer dialogue histories. As shown in Figure 3(a), Ours outperforms the Baseline across all turns, and the performance gap becomes larger in later turns. While the overall improvement is +4.13 points, the gain is only +1.29 points at Turn 2 but increases to +7.09 points at Turn 6. This shows that the benefit of Ours is not limited to average performance improvement, but becomes more pronounced when the dialogue becomes longer and the model must consider more previous context.

Figure 3(b) further shows that this difficulty in later turns is related to reference distance. When Turn 6 examples are grouped by the referenced turn, both the Baseline and Ours tend to show lower performance when the query refers to an older turn. In other words, when the current query refers not to a recent previous turn but to information from earlier in the dialogue, the model must identify the correct antecedent from a longer history, making API prediction more difficult.

Nevertheless, READ remains stronger across all reference distances, suggesting that explicit rewriting helps expose long-range antecedents before

Family	Model	Accuracy			Latency		
		Baseline	READ	Δ	Baseline	READ	READ/Baseline
Qwen3	0.6B	71.9	73.4	+1.5	1.00×	1.20×	1.20×
	1.7B	73.4	76.8	+3.4	2.85×	3.31×	1.16×
	4B	73.6	77.5	+3.9	7.31×	8.26×	1.13×
GLM-edge	1.5B	69.3	74.3	+5.0	1.00×	1.11×	1.11×
	4B	74.5	77.0	+2.5	2.07×	2.31×	1.12×

Table 4: Relative on-device accuracy–latency comparison on a Samsung Galaxy Tab S11. Latency is normalized within each model family using the smallest Baseline model as 1.00×. The READ/Baseline column reports the relative latency overhead of READ compared with the same-size Baseline.

API prediction. We provide additional success and failure case analyses in Appendix E, further illustrating when explicit rewriting helps and where residual errors remain.

Robustness on Realistic Validation Set. The main experiments above use a controlled synthetic benchmark to isolate context-resolution effects under known API annotations. To examine whether the same advantage transfers beyond generated utterances, we construct a separate 1,000-example human-written realistic validation set. The set was used only for evaluation; details of its construction and validation are provided in Appendix C.3.

As shown in Table 3, READ improves over the Baseline on the Realistic Validation Set, with a 6.1-point average gain. Compared with the corresponding synthetic benchmark results in Table 2 for the same three models, both methods show lower accuracy, confirming that the realistic set is more challenging. Moreover, READ shows a smaller performance drop than the Baseline (-2.1 vs. -3.2 points), indicating that the benefit of explicit context resolution is not limited to the generated benchmark and remains visible under noisier human-written follow-up requests.

4.3 Efficient Alternative to Model Scaling

Table 4 reports the on-device accuracy–latency trade-off measured on a Samsung Galaxy Tab S11. The key observation is that READ shifts the accuracy–latency frontier: smaller READ models often approach or exceed the Baseline performance of larger models while requiring substantially lower relative latency. For example, Qwen3-1.7B with READ outperforms the Qwen3-4B Baseline while requiring less than half of its normalized latency. GLM-edge shows a similar trend, with the 1.5B READ model nearly matching the 4B Baseline at lower relative latency.

These results suggest that READ provides a more efficient path to improving on-device performance than simply increasing model size. Although READ introduces additional latency by generating both the rewritten query and the API prediction, this overhead is limited compared with the latency increase from moving to a larger model. Further details on the Android evaluator, GGUF quantization, latency measurement protocol, and cache handling are provided in Appendix F.

4.4 Generalization Beyond the In-Domain Benchmark

We evaluate the generalization performance of the proposed method along two complementary axes. The first is Broad API-planning transfer, which examines general function-calling ability on external benchmarks. The second is Rewrite-sensitive transfer, which evaluates the effect of explicit rewriting in unseen multi-turn settings involving implicit references.

First, to evaluate broad API-planning transfer, we assess the Phi-4 (4B) model with READ on MetaTool and DroidCall. As shown in Table 5, the proposed model remains competitive with existing open models and strong baselines, although it does not outperform all models on every benchmark. These results suggest that READ-trained Phi-4 remains competitive on external API-planning benchmarks.

For rewrite-sensitive transfer, we further evaluate READ on the HammerBench External subset (turn > 1), where accurate prediction requires resolving implicit references across turns.

Table 6 presents the Final Call Exact Match results on HammerBench External, where HB Prompt denotes the original benchmark prompt. For Qwen3 and Phi-4, 'Ours' denotes the fine-tuned READ model trained with the single-pass target format. For closed-source models, READ denotes a prompt-only rewrite-before-plan formu-

Model	MetaTool	DroidCall
LLaMA-3 (3B)	89.64	82.00
Qwen2.5 (3B)	88.29	83.00
GPT-4o-mini	88.31	71.50
Phi-4 (4B) (Ours)	93.07	79.10

Table 5: Accuracy (%) on MetaTool and DroidCall. Baselines are from benchmark reports; Phi-4 (Ours) follows the official protocol.

Model	**HB Prompt**	**Baseline**	**READ**	**Δ**
Sonnet-4.6	40.25	44.12	48.03	+3.91
Haiku-4.5	36.78	39.85	43.27	+3.42
Gemini-3.1	38.64	41.76	45.18	+3.42
GPT-5.4	38.92	42.37	46.67	+4.30
Qwen3 (4B)	54.69	53.30	59.75	+6.45
Phi-4 (4B)	51.15	52.07	54.69	+2.62
Avg	43.41	45.58	49.60	+4.02

Table 6: Final Call Exact Match (%) on HammerBench External (multi-turn, turn > 1). Closed-source models use prompt-only rewrite-before-plan inference, while Qwen3 and Phi-4 use fine-tuned READ. Δ = READ–Baseline.

lation without parameter updates. We therefore interpret the two groups separately: the closed-source results test whether the rewrite-before-plan interface is useful as an inference-time prompting strategy, while the open-model results test the full fine-tuned READ framework.

These results suggest that rewrite-before-plan can transfer beyond the in-domain benchmark.

5 Ablation Studies

5.1 Effect of Explicit Decomposition and Autoregressive Ordering

To disentangle the source of READ's improvement, we compare three output formulations: a history-based baseline that directly predicts the API plan and arguments, a reverse-order ablation that predicts the plan and arguments before the rewritten query, and READ, which generates the rewritten query before API prediction. The reverse-order ablation uses the same rewrite supervision as READ, but prevents the rewrite from serving as a prefix condition for planning.

As shown in Table 7, the results exhibit a stepwise trend: average accuracy increases from 70.5% for the baseline to 74.1% for the reverse-order ablation, and further to 74.7% for READ. A paired bootstrap analysis shows that the reverse-order variant remains significantly better than the Baseline

Model	Baseline	Reverse-order	READ
Qwen3 (4B)	73.6	77.1	77.5
Phi-4 (4B)	67.0	68.9	71.0
Llama-3.2 (3B)	72.2	74.2	74.8
SmolLM3 (3B)	67.2	75.2	74.9
Granite3.3 (2B)	73.9	75.6	75.7
GLM-edge (1.5B)	69.3	73.4	74.3
Avg	70.5	74.1	74.7

Table 7: Effect of output ordering on API-call exact match. Reverse-order predicts the API plan before the rewrite.

by +3.6 points, while still underperforming READ on average (-0.6 points, 95% CI [-1.2, -0.1]). These results suggest that explicit rewrite supervision is the main source of improvement, while rewrite-before-plan ordering provides a modest additional benefit beyond rewrite supervision.

6 Discussion and Future Directions

The remaining gap between READ and Oracle Rewrite suggests that rewrite quality remains an important bottleneck. Although explicit rewriting improves context-dependent API prediction, incorrect, incomplete, or semantically shifted rewrites can still propagate to the final API call. Future work could explore stronger rewrite supervision, lightweight verification, or faithfulness-aware objectives to reduce unsupported rewrite changes without substantial on-device overhead.

Another promising direction is selective rewriting. READ rewrites every user turn, even when the utterance is already self-contained. A lightweight mechanism for deciding whether context resolution is necessary could avoid unnecessary decoding, reducing latency and inference cost while preserving the accuracy benefits of explicit rewriting.

7 Conclusion

We proposed READ, an explicit decomposition approach for on-device multi-turn mobile API dialogue. Experiments across 4B-class and smaller models show that READ consistently improves API-call exact-match accuracy over the baseline and alternative comparison settings. Additional evaluations on realistic examples and external function-calling benchmarks demonstrate READ's robustness and generalization, while mobile-device latency measurements show its practical accuracy–latency trade-off for on-device multi-turn API planning.

Limitations

Potential Risks from Erroneous API Calls. Incorrect API predictions in mobile environments may lead to unintended actions, such as sending messages to the wrong recipient or executing sensitive operations without sufficient confirmation. Practical deployments should therefore include user confirmation, permission checks, and safeguards for sensitive API calls.

Synthetic Data and Real-World Coverage. Our main benchmark is generated through a controlled data-construction pipeline and validated through automatic filtering and human assessment. While this design enables systematic control over reference distance, API labels, and rewrite supervision, it cannot fully capture the distribution of real production mobile-assistant logs. We partially address this limitation with a separate human-written realistic validation set, but future work should evaluate READ on privacy-preserving real-world interaction logs or user-consented deployment traces.

References

Apple Inc. 2024. Use siri on all your apple devices. https://support.apple.com/en-us/105020. Accessed: 2026-05-23.

Haodong Duan, Jueqi Wei, Chonghua Wang, Hongwei Liu, Yixiao Fang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. BotChat: Evaluating LLMs' capabilities of having multi-turn dialogues. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3184–3200, Mexico City, Mexico. Association for Computational Linguistics.

Google. 2025. Common intents | android developers. https://developer.android.com/guide/components/intents-common. Accessed: 2026-05-23.

Google DeepMind. 2025. Gemini 2.0 flash model card. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash. Release date: 2025-02-05; Accessed: 2026-05-23.

Ming Gu and Yan Yang. 2024. Plan, generate and complicate: Improving low-resource dialogue state tracking via easy-to-difficult zero-shot data augmentation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6988–7005, Bangkok, Thailand. Association for Computational Linguistics.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*.

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

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations*.

Hugging Face. 2025. Smollm3-3b model card. https://huggingface.co/HuggingFaceTB/SmolLM3-3B. Accessed: 2026-05-23.

Erik Johannes Husom, Arda Goknil, Merve Astekin, Lwin Khin Shar, Andre Kåsen, Sagar Sen, Benedikt Andreas Mithassel, and Ahmet Soylu. 2025. Sustainable llm inference for edge ai: Evaluating quantized llms for energy efficiency, output accuracy, and inference latency. *Preprint*, arXiv:2504.03360.

IBM. 2025. Granite-3.3-2b-instruct model card. https://huggingface.co/ibm-granite/granite-3.3-2b-instruct. Accessed: 2026-05-23.

Ivica Kostric and Krisztian Balog. 2024. A surprisingly simple yet effective multi-query rewriting method for conversational passage retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pages 2271–2275, Washington, DC, USA. Association for Computing Machinery.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Haoyi Li, Angela Yifei Yuan, Soyeon Caren Han, and Christopher Leckie. 2025. Spade: Structured prompting augmentation for dialogue enhancement in machine-generated text detection. *Preprint*, arXiv:2503.15044.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Xiang Li, Zhenyan Lu, Dongqi Cai, Xiao Ma, and Mengwei Xu. 2024. Large language models on mobile devices: Measurements, analysis, and insights. In *EdgeFM@MobiSys*, pages 1–6.

Jiarui Lu, Bo-Hsiang Tseng, Joel Ruben Antony Moniz, Site Li, Xueyun Zhu, Hong Yu, and Murat Akbacak. 2023. 5ider: Unified query rewriting for steering, intent carryover, disfluencies, entity carryover and repair. In *INTERSPEECH 2023*, pages 760–764. ISCA.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore. Association for Computational Linguistics.

Meta. 2024. Llama 3.2 model card. https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2/. Accessed: 2026-05-23.

Microsoft, Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, Hany Awadalla, Nguyen Bach, Jianmin Bao, Alon Benhaim, Martin Cai, Vishrav Chaudhary, Congcong Chen, Dong Chen, Dongdong Chen, Junkun Chen, Weizhu Chen, Yen-Chun Chen, Yiling Chen, Qi Dai, Xiyang Dai, and 56 others. 2025. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *arXiv preprint arXiv:2503.01743*.

Fengran Mo, Kelong Mao, Yutao Zhu, Yihong Wu, Kaiyu Huang, and Jian-Yun Nie. 2023. ConvGQR: Generative query reformulation for conversational search. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4998–5012, Toronto, Canada. Association for Computational Linguistics.

OpenAI. 2025a. Introducing openai gpt-4.1. https://openai.com/index/gpt-4-1/. Published: 2025-4-14; Accessed: 2026-05-23.

OpenAI. 2025b. Introducing openai gpt-5. https://openai.com/index/introducing-gpt-5/. Published: 2025-08-07; Accessed: 2026-05-23.

OpenAI. 2025c. Introducing openai o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/. Published: 2025-04-16; Accessed: 2026-05-23.

Shalin Parikh, Dharmin Dave, Reema Patel, and Nishant Doshi. 2019. Security and privacy issues in cloud, fog and edge computing. *Procedia Computer Science*, 160:734–739. The 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops.

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

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Samsung Electronics. 2025. Bixby user guide. https://www.samsung.com/us/apps/bixby/bixby-user-guide/. Accessed: 2026-05-23.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, Singapore. Association for Computational Linguistics.

Haiyang Shen, Yue Li, Desong Meng, Dongqi Cai, Sheng Qi, Li Zhang, Mengwei Xu, and Yun Ma. 2025. Shortcutsbench: A large-scale real-world benchmark for API-based agents. In *The Thirteenth International Conference on Learning Representations*.

Zhengliang Shi, Shuo Zhang, Weiwei Sun, Shen Gao, Pengjie Ren, Zhumin Chen, and Zhaochun Ren. 2024. Generate-then-ground in retrieval-augmented generation for multi-hop question answering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7339–7353, Bangkok, Thailand. Association for Computational Linguistics.

Joe Stacey, Jianpeng Cheng, John Torr, Tristan Guigue, Joris Driesen, Alexandru Coca, Mark Gaynor, and Anders Johannsen. 2024. LUCID: LLM-generated utterances for complex and interesting dialogues. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*, pages 56–74, Mexico City, Mexico. Association for Computational Linguistics.

THUDM. 2024. Glm-edge: Models for edge devices. https://github.com/THUDM/GLM-Edge. Accessed: 2026-05-23.

Jun Wang, Jiamu Zhou, Muning Wen, Xiaoyun Mo, Haoyu Zhang, Qiqiang Lin, Cheng Jin, Xihuai Wang, Weinan Zhang, Qiuying Peng, and Jun Wang. 2025. Hammerbench: Fine-grained function-calling evaluation in real mobile device scenarios. *Preprint*, arXiv:2412.16516.

Zhuoer Wang, Leonardo F. R. Ribeiro, Alexandros Papangelis, Rohan Mukherjee, Tzu-Yen Wang, Xinyan

Zhao, Arijit Biswas, James Caverlee, and Angeliki Metallinou. 2024. FANTAstic SEquences and where to find them: Faithful and efficient API call generation through state-tracked constrained decoding and reranking. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6179–6191, Miami, Florida, USA. Association for Computational Linguistics.

Jie Xiao, Qianyi Huang, Xu Chen, and Chen Tian. 2025. Understanding large language models in your pockets: Performance study on cots mobile devices. *Preprint*, arXiv:2410.03613.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. *Preprint*, arXiv:2309.07597.

Weikai Xie, Li Zhang, Shihe Wang, Rongjie Yi, and Mengwei Xu. 2024. Droidcall: A dataset for llm-powered android intent invocation. *Preprint*, arXiv:2412.00402.

Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. 2023a. Baize: An open-source chat model with parameter-efficient tuning on self-chat data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6268–6278, Singapore. Association for Computational Linguistics.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023b. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *Preprint*, arXiv:2312.12148.

Xiao Yan and Yi Ding. 2025. Are we there yet? a measurement study of efficiency for llm applications on mobile devices. *Preprint*, arXiv:2504.00002.

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

Fanghua Ye, Meng Fang, Shenghui Li, and Emine Yilmaz. 2023. Enhancing conversational search: Large language model-aided informative query rewriting. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Yue Zhou, Chenlu Guo, Xu Wang, Yi Chang, and Yuan Wu. 2024. A survey on data augmentation in large model era. *Preprint*, arXiv:2401.15422.

Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762.

A Reproducibility Resources

We provide reproducibility resources at the following repositories. The main code repository contains the training, evaluation, data construction, filtering, and result-analysis code: `https://github.com/paperread153/mobile-api-dialogue-repro`

The model-adapter repository contains the LoRA adapter checkpoints and inference configuration files: `https://huggingface.co/anon153/mobile-api-dialogue-adapters`

The on-device evaluator repository contains the Android evaluation app, llama.cpp integration, GGUF conversion instructions, and latency measurement protocol: `https://github.com/paperread153/ondevice-api-evaluator`

B Dataset Construction Details

In this appendix, we describe the Full list of APIs, API Descriptions JSON and the dialogue generation process used in constructing our dataset.

B.1 API List and Types

Table 8 is the list of 41 APIs supported in our framework.

B.2 API Descriptions JSON

The API Descriptions utilized in dataset construction consist of a total of 41 APIs, designed to reflect realistic user requests within mobile environments. Each API is commonly defined in a JSON format with the following six main keys:

- **plan**: The name of the API specifying the specific action the system performs in response to user requests.
- **description**: Provides detailed explanations of the purpose and functions of the API, helping the system accurately understand API behavior and select appropriate APIs based on context.
- **arguments**: Specifies parameter information required for API calls, clearly distinguishing each parameter by name, description, data type, and whether it is required.
- **returns**: Provides reference information for generating user responses after API calls. This includes the format and specific examples of response messages.

#	API
1	ACTION_EDIT_ALARM
2	ACTION_EDIT_CONTACT
3	ACTION_EDIT_DOCUMENT
4	ACTION_EDIT_VIDEO
5	ACTION_INSERT_CONTACT
6	ACTION_INSERT_EVENT
7	ACTION_NAVIGATE_TO_LOCATION
8	ACTION_OPEN_CONTENT
9	dial
10	play_music
11	play_video
12	search_location
13	send_email
14	send_message
15	ACTION_CREATE_DOCUMENT
16	ACTION_OPEN_DOCUMENT
17	ACTION_GET_CONTENT
18	ACTION_GET_RINGTONE
19	ACTION_IMAGE_CAPTURE
20	ACTION_PICK
21	ACTION_SET_ALARM
22	ACTION_SET_TIMER
23	ACTION_SHOW_ALARMS
24	ACTION_VIDEO_CAPTURE
25	ACTION_VIEW_CONTACT
26	INTENT_ACTION_STILL_IMAGE_CAMERA
27	INTENT_ACTION_VIDEO_CAMERA
28	get_contact_info
29	get_contact_info_from_uri
30	open_settings
31	web_search
32	launch_application
33	ACTION_SET_RINGTONE
34	ACTION_VIEW_EVENT
35	ACTION_SHOW_TIMERS
36	ACTION_VIEW_CALL_LOG
37	ACTION_VIEW_SENT_EMAILS
38	ACTION_VIEW_SENT_MESSAGES
39	ACTION_VIEW_WEB_HISTORY
40	ACTION_VIEW_PLAYLIST
41	ACTION_VIEW_RECENT_APPS

Table 8: Full list of APIs

- **examples**: Includes specific examples of API calls annotated through API planning annotations based on actual user queries, clearly illustrating scenarios in which the API is invoked.

- **next_turn_plans**: Lists potential APIs callable in subsequent turns following the current API invocation. This section includes the possible next APIs along with reasons and concrete examples, facilitating natural conversational flow in multi-turn interactions.

Listing 1 below demonstrates an actual API example, showing JSON data for the 'dial' API, which performs phone-calling functions. These API Descriptions are leveraged at each stage of dataset creation as required and are also used during SLM LoRA SFT training and inference processes.

B.3 Dataset Generation Process

The following outlines the procedures and prompts used during the dataset generation process. Each dialogue is generated sequentially through the following four steps: **Step 1: Assistant Response Generation (Listing 2)** The system generates an appropriate response based on the user's initial query. **Step 2: Next User Query Generation (Listing 3, Listing 4)** A natural follow-up user query is generated based on the assistant's response. **Step 3: Query Rewriting (Listing 5)** The user query is rewritten into a clear, single sentence, reflecting the context of the previous turn. **Step 4: API Call and Parameter Annotation (Listing 6)** Based on the rewritten query, the system annotates and specifies the API and parameters to be invoked. This procedure completes one turn of interaction, alternating between user and assistant utterances, and explicitly specifies the API action sequence to be executed in each turn. In this study, we iterated this four-step process from a minimum of one turn to a maximum of five turns, creating a dataset of approximately 17,000 multi-turn dialogue samples through five iterations.

```
{
 "plan": "dial",
 "description": "Initiates a phone call to the given number.\n\nThis function triggers the phone app to call the specified phone number. It requires the CALL_PHONE permission if used within an app.",
 "arguments": {
  "phone_number": {
   "description": "The phone number to dial.",
   "type": "str",
   "required": true
  }
 },
 "returns": {
  "description": "A confirmation message indicating that the call is being initiated to the specified phone number.",
  "type": "str",
  "example": [
   "Dialing 010-1234-5678..."
  ]
 },
 "examples": [
  "# Dial a phone number\ndial(\"123-456-7890\")"
 ],
 "next_turn_plans": [
 {
  "plan": "send_message",
  "reason": "If the call was not answered or
```

```
          could not be completed, it can
         naturally lead to sending a text
         message to the same number.",
   "example": {
    "query": "Call 010-1111-2222",
    "device_response": "Dialing 010-1111-2222
        ...",
    "next_turn_query": "Send him a text
        instead saying 'I'm on my way.'",
    "rewritten_query": "Send 010-1111-2222 a
        text instead saying 'I'm on my way.'",

    "next_turn_plan": {
     "plan": "send_message",
      "arguments": {
       "to": "010-1111-2222",
       "message": "I'm on my way."
      }
     }
    }
  },
  {
   "plan": "ACTION_INSERT_CONTACT",
   "reason": "If the number dialed is not in
        the contacts, the user can add it as a
         new contact after the call.",
   "example": {
    "query": "Dial 02-123-4567",
    "device_response": "Calling 02-123-4567",
    "next_turn_query": "Save this number",
    "rewritten_query": "Save 02-123-4567",
    "next_turn_plan": {
     "plan": "ACTION_INSERT_CONTACT",
     "arguments": {
      "contact_info": {
      "phone": "02-123-4567"
      }
     }
    }
   }
  },
  {
   "plan": "ACTION_VIEW_CALL_LOG",
   "reason": "This scenario is when a user
        has made a phone call and then wants
        to see if it went through or check
        their recent call list. For example,
        right after dialing a call with dial
        existing API, the user can ask to see
        their call log, and
        ACTION_VIEW_CALL_LOG will display it.
        Saying Call Mom followed by Show my
        call log is a natural sequence.",
   "example": {
    "query": "Call 010-1234-5678",
    "device_response": "Dialing Mom at 010-12
        34-5678...",
    "next_turn_query": "Show my call log",
    "rewritten_query": "Show my call log",
    "next_turn_plan": {
     "plan": "ACTION_VIEW_CALL_LOG",
     "arguments": {}
    }
   }
  }]
 }
```

Listing 1: Example JSON Data

```
After successfully executing a plan based on
     the user's query, this will help
    generate natural and accurate
    device_responses for the mobile AI
    assistant. You will receive detailed
    tool descriptions (plans), results from
    plan execution, previous conversation
    history, and the user's queries. Your
    task is to explicitly create clear,
    contextually appropriate
    device_responses based on these inputs.

Carefully follow the instructions below:
 1. Diversity and Successful device_response
      (Brief): Utilize diverse vocabulary
     and sentence structures to maintain
     naturalness with varying tones and
     attitudes, yet keep concise.
 2. Contextual Accuracy: Fully understand
     the provided tool description and user
     query to generate contextually accurate
      device_responses.
 3. When generating a device_response for
     tools like VIEW_SENT_SOMETHINGS that
     check past records, ensure the
     reliability of the data by referencing
     the conversation_history that contains
     past device_response records.
 4. Please refer to the 'returns' section in
      the tool description for the format of
      the content to be written in
     device_response. Based on the 'returns'
      section, generate a variety of
     examples in a similar format. If
     conversation_history exists, you may
     also need to refer to previous
     responses.
 5. If the return value is not explicitly
     stated in the user query but needs to
     be generated creatively, it should be
     produced in a diverse and non-
     repetitive manner.
 6. In addition, make sure to explicitly
     generate device_responses that clearly
     reflect the values returned by the
     executed plan. If the returned values
     include specific details such as URIs
     or phone numbers, state them clearly
     and accurately.

Below is the provided tool (plan) including
    the plan and execution results:
Tool: {tool}

When generating a device_response based on
    the tool, each conversation_history, and
     query, imagine yourself as a Mobile
    Device. Also, make sure to vary the tone
     and phrasing for each generated
    response.
Based on the provided query-device_response
    sets below, clearly and appropriately
    add a 'device_response' field.
Do not modify any of the values provided in
    the query-device_response sets. Please
    add a new key "device_response" with an
    appropriate value, directly to the top-
    level of the JSON object, not inside any
     nested field
{dataset}
```

Listing 2: Step1 prompt for Assistant Response

```
Below are the user's previous questions and the system's responses in [previous_turn_datas], along with a description of the next possible action in [description].
Based on [previous_turn_datas] and [description], you need to write a natural and clear follow-up request from the user's perspective, named next_turn_query and explicitly rewrite this as rewritten_query.

Guidelines for writing next_turn_query:
 1. Clearly understand the content of the previous turn (query and device_response) and create a natural follow-up request as a human would that fits the context.
 2. The next_turn_query should be as concise as possible and can be written in various forms, such as commands, questions, or asking with detailed descriptions. However, explanations about the next_turn_query themselves should not be provided.
 3. The generated next_turn_query, based on the query and device_response, should comprehensively cover all possible use cases of the tool, including various parameters to verify the tool's ability to handle different types of tasks. Information for parameters with required=True must be included.
 4. The generated next_turn_query should be solvable using the given tools. if not, next_turn_query should be blank value like ""
 5. Use the provided [description] explanations (especially refer to the "reason" key for the next-turn query) to naturally and creatively express requests that the user might realistically make.
 6. When objects or return values are included in the 'query' and 'device_response', you should utilize them only when they represent parameters of the provided tool. but avoid explicitly mentioning their values directly. Instead, refer to them using pronouns such as "it," "that," "there," "her," or "him.".
 7. Use pronouns to refer to one or more of the provided tool's parameters by utilizing conversation_history, query or device_response.
 8. Each next_turn_query should have only one purpose.
 9. Do not generate the same next_turn_query among the provided data.

Guidelines for writing rewritten_query:
The purpose of the rewritten_query is to convert the next_turn_query into a clear sentence.
 1. Replace ambiguous pronouns used in next_turn_query (e.g., "it," "that," "there") with explicit information (query or device_response) from the previous turn. do not change sentence format against multi_turn_query
 2. explicit values in rewritten_query must be taken from query, device_response, next_turn_query
 3. Keep the structure and form of the next_turn_query and sentence as they are, but replace pronouns with explicit values only when the pronouns refer to conversation_history, query, or device_response.

Again, you need to consider important rules below.
rule 1. next_turn_query covers required parameter in description using pronouns
rule 2. If conversation_history, query, and device_response include explicit values representing the provided tool parameters, then next_turn_query should contain pronouns.
rule 3. if pronouns represent provided tool's parameter, rewritten_query is same format sentence with next_turn_query, but pronouns
rule 4. explicit values in rewritten_query must be taken from query, device_response, next_turn_query

[description]
{description}

[example]
{example}

Without modifying the JSON data below, add the next_turn_query, rewritten_query with an appropriate value, directly to the top-level of the JSON object, not inside any nested field. Then generate the entire JSON data for all. Please generate next_turn_query as diversely as possible and it should align with the intended purpose of the tool and be something that a real user would actually say or use.

[previous_turn_datas]
{previous_turn_data}
```

Listing 3: Step2 prompt for Next Turn Query

```
Below are the user's previous questions and the system's responses in [previous_turn_datas], along with a description of the next possible action in [description].
Based on [previous_turn_datas] and [description], you need to write a natural and clear follow-up request from the user's perspective, named next_turn_query and explicitly rewrite this as rewritten_query.

Guidelines for writing next_turn_query:
 1. Clearly understand the content of the previous turn (query and
```

```
        device_response) and create a natural
        follow-up request as a human would that
         fits the context.
  2. The next_turn_query should be as
        complicated as possible and can be
        written in various forms, such as
        commands, questions, or asking with
        detailed descriptions. However,
        explanations about the next_turn_query
        themselves should not be provided.
  3. It should sound as if a real person is
        talking to a device include natural
        fillers, hesitations, or even
        unnecessary remarks to make it feel
        more human.
  4. The generated next_turn_query, based on
        the query and device_response, should
        comprehensively cover all possible use
        cases of the tool, including various
        parameters to verify the tool's ability
         to handle different types of tasks.
        Information for parameters with
        required=True must be included.
  5. The generated next_turn_query should be
        solvable using the given tools. if not,
         next_turn_query should be blank value
        like ""
  6. Use the provided [description]
        explanations (especially refer to the "
        reason" key for the next-turn query) to
         naturally and creatively express
        requests that the user might
        realistically make.
  7. When objects or return values are
        included in the 'query' and '
        device_response', you should utilize
        them only when they represent
        parameters of the provided tool. but
        avoid explicitly mentioning their
        values directly. Instead, refer to them
         using pronouns such as "it," "that," "
        there," "her," or "him.".
  8. Use pronouns to refer to one or more of
        the provided tool's parameters by
        utilizing conversation_history, query
        or device_response.
  9. Each next_turn_query should have only
        one purpose.
  10. Do not generate the same
        next_turn_query among the provided data.


Guidelines for writing rewritten_query:
The purpose of the rewritten_query is to
     convert the next_turn_query into a clear
        sentence.
  1. Replace ambiguous pronouns used in
        next_turn_query (e.g., "it," "that," "
        there") with explicit information (
        query or device_response) from the
        previous turn. do not change sentence
        format against multi_turn_query
  2. explicit values in rewritten_query must
        be taken from query, device_response,
        next_turn_query
  3. Keep the structure and form of the
        next_turn_query and sentence as they
        are, but replace pronouns with explicit
         values only when the pronouns refer to
         conversation_history, query, or
         device_response.

Again, you need to consider important rules
     below.
rule 1. next_turn_query covers required
     parameter in description using pronouns
rule 2. If conversation_history, query, and
     device_response include explicit values
     representing the provided tool
     parameters, then next_turn_query should
     contain pronouns.
rule 3. if pronouns represent provided tool'
     s parameter, rewritten_query is same
     format sentence with next_turn_query,
     but pronouns
rule 4. explicit values in rewritten_query
     must be taken from query,
     device_response, next_turn_query

[description]
{description}

[example]
{example}

Without modifying the JSON data below, add
     the next_turn_query, rewritten_query
     with an appropriate value, directly to
     the top-level of the JSON object, not
     inside any nested field. Then generate
     the entire JSON data for all. Please
     generate next_turn_query as diversely as
      possible and it should align with the
     intended purpose of the tool and be
     something that a real user would
     actually say or use.

[previous_turn_datas]
{previous_turn_data}
```

Listing 4: Step2 prompt for coreference Next Turn Query

```
Please generate a rewritten_query based on
     the provided conversation_history and
     query.

The goal of the rewritten_query is to
     rewrite the original query into a clear,
      explicit, and fully self-contained
     sentence by replacing ambiguous pronouns
     , demonstratives, or referential
     expressions (such as "it," "that," "
     there") with explicit information
     directly from the conversation_history.

Important Guidelines:
Rule 1: Identify all ambiguous pronouns,
     demonstratives, or referential terms in
     the query and replace each of them
     explicitly with corresponding details
     from the conversation_history.
Rule 2: Keep exactly the same sentence
     structure as the original query. Only
     replace ambiguous references with
     explicit values. However, exclude any
     natural fillers, hesitations, or
     conversational asides when rewriting the
      sentence.
```

```
Rule 3: If the query references specific values such as URIs, file paths, phone numbers, email addresses, or physical addresses, you MUST explicitly include these values in the rewritten_query. Every explicit value used in the rewritten_query must appear exactly as presented in the conversation_history or query. Do NOT alter these values in any way, and do NOT generate or include any additional information that is not explicitly provided.
Rule 4: Do NOT generate or include any information, words, or values outside of the provided conversation_history and query. No additional context, explanation, or commentary should be added.

Here are clear and correct examples demonstrating the desired format and behavior:
examples: {examples}

Now, using only the data provided below, generate the rewritten_query. Respond strictly following the JSON format given below and include no additional text or explanation:
{{"rewritten_query": ""}}
data:
{data}
```

Listing 5: Step3 prompt for rewrite

```
Please help me generate a plan-calling dataset. I will provide a tool description and some example data.
Based on this tool, you need to generate answers for the provided queries. In other words, you need to provide answers that call the tool to resolve the user's query. Here are my requirements:

1. For the generated queries, you must provide answers using the tool. That is, specify the tool used and the values for each parameter.
2. When providing parameters, if a parameter has required=False, it is not necessary to provide its value. However, if the query includes that parameter, you must provide it.
3. The generated data must be presented in the format given in the example above.
4. The parameter values generated with plan calls must be values that can be inferred from the user's query. You must not fabricate parameters that cannot be obtained from the user's request.
5. If a query does not include a required=True parameter, set the answer.plan value as 'N/A' and answer.arguments.reason with reasons"

Below are the tool I provided and some examples of query-answer pairs:
Tool: {tool}
Examples: {examples}

Now, please help me generate answer pairs for the queries below. The generated data should include both a query and an answer key.
The results must be generated in JSON format like the example above.
Do not fabricate parameters for the tool; only use parameters that can be inferred from the user's request.
{data}
```

Listing 6: Step4 prompt for annotation

C Dataset Filtering Details

This appendix provides detailed descriptions of the data quality validation procedures and evaluation criteria, both automated and human, that were employed during the construction of the multi-turn dialogue dataset.

C.1 Evaluation Subjects and Data Filtering Process

To ensure dataset quality, data generated by OpenAI's o4-mini, GPT-5-mini, GPT-4.1, and Google's Gemini-2.0-Flash models underwent an initial automated evaluation, followed by additional human evaluation for reliability verification.

Automated Evaluation (LLM-based Filtering) Data incorrectly generated during model-specific generation steps (Steps 2-4) were automatically removed according to predefined evaluation criteria. Additionally, following OpenAI and Gemini content policies, any personally identifiable or potentially offensive content was automatically excluded. At the final step of each turn, data containing incorrect API and parameter annotations, as well as inconsistencies in language, were filtered out.

Human Evaluation Data passing the automated evaluation was reviewed directly by evaluators. A total of 164 randomly sampled dialogues per model across turns 2-5 and different APIs were assessed to confirm the reliability of the automated evaluation results.

C.2 Detailed Evaluation Criteria

Five criteria applied in both automated and human evaluations are detailed below:

- **Criterion 1: Proper Reference to Previous Dialogue History**
  Generated utterances were evaluated for accurate referencing of information from prior turns. Data with unclear or incorrect references were filtered out.

- **Criterion 2: Alignment of Generated Queries with Target APIs**
  Queries were assessed for alignment with the purpose of the target APIs. Data not aligning with the intended API purpose were removed.

- **Criterion 3: Clear Query Rewriting According to Intent**
  Ambiguous pronouns or instructions were clearly rewritten, and the clarity of rewritten queries was evaluated. Unclear rewrites were filtered out.

- **Criterion 4: Maintenance of Linguistic Consistency**
  Data were evaluated for naturalness, coherence, and consistency. Utterances containing unnatural expressions, repeated phrases, or grammatical errors were excluded.

- **Criterion 5: Accuracy of API and parameter Annotations**
  Data annotations were checked against API descriptors (API Descriptor JSON) for accuracy, removing data with incorrect annotations.

C.3 Evaluation Results

Automated Evaluation (LLM-based Filtering) As shown in Table 1, the o4-mini had the lowest dialogue-quality filtering rate and the lowest overall/average filtering rate; GPT-5-mini had the lowest annotation-quality filtering rate.

Generator-Selection Human Assessment. As an initial human assessment for generator selection, we evaluated randomly sampled data from each candidate generator. Specifically, 164 samples per model were reviewed using the same five criteria as the automated evaluation. The results showed that o4-mini achieved the highest pass rate at 95.1%, followed by GPT-5-mini at 92.5%, GPT-4.1 at 90.9%, and Gemini-2.0-Flash at 90.2%. These results support the selection of o4-mini as the final data generator among the tested candidates.

Human Evaluation Protocol. To further validate the quality of the final synthetic dataset, we conducted an additional human evaluation on 2,000 instances sampled from the final filtered dataset. The samples were selected using stratified sampling over dialogue turns and API categories to ensure coverage of both short-context and long-context examples. The evaluation was conducted by 10 trained evaluators: three developers with experience in mobile assistant or API-based systems and seven graduate students familiar with NLP or dialogue annotation. All evaluators and developers involved in the human evaluation and realistic validation set construction were informed that their annotations and written examples would be used for research evaluation and dataset validation, and they voluntarily agreed to participate.

Each evaluator was provided with the dialogue history, the current user query, the rewritten query, the API descriptor, and the annotated API call with arguments. Evaluators judged each instance using three binary criteria:

- **API Alignment:** whether the selected API plan is consistent with the user query and the API descriptor.

- **Reference Resolution:** whether the rewritten query correctly resolves pronouns, ellipsis, or implicit references using the dialogue history.

- **Argument Groundability:** whether all required API arguments can be directly inferred from the query, rewritten query, or dialogue history without introducing unsupported information.

For each criterion, evaluators marked the instance as either valid or invalid. If an instance was marked invalid, evaluators additionally selected one or more error types from a predefined list, including wrong API, incorrect reference target, missing required argument, incorrect argument value, hallucinated argument value, and unnatural or ambiguous rewrite.

To estimate annotation reliability, 300 instances were double-annotated by two independent evaluators. Disagreements were resolved by an expert adjudicator after reviewing the dialogue history, rewritten query, API descriptor, and annotated API call. Figure 4 shows the annotation interface and the instructions presented to evaluators, including the three binary criteria and the instruction not to assume or invent missing information.

Human Evaluation Results. Table 9 summarizes the human evaluation results. Overall, 93.8% of the sampled instances passed all three criteria. Among the individual criteria, API Alignment achieved the highest pass rate at 96.4%, followed by Reference Resolution at 94.7% and Argument Groundability at 95.2%. The agreement rate on

Criterion	Pass Rate (%)	Fail Rate (%)
API Alignment	96.4	3.6
Reference Resolution	94.7	5.3
Argument Groundability	95.2	4.8
All Criteria Passed	93.8	6.2

Table 9: Human evaluation results on 2,000 stratified samples from the final dataset. An instance is counted as passing all criteria only when it satisfies API alignment, reference resolution, and argument groundability.

Error Type	Invalid Cases (%)	Full Sample (%)
Reference target error	34.7	2.15
Weakly grounded argument	26.6	1.65
Wrong API selection	14.5	0.90
Incorrect argument value	12.9	0.80
Hallucinated argument value	11.3	0.70

Table 10: Error type distribution among invalid human-evaluation instances. Invalid Cases reports percentages within invalid instances, while Full Sample reports percentages over all 2,000 evaluated samples.

the double-annotated subset was 91.6%, with a Cohen's κ of 0.82, indicating substantial agreement among evaluators.

Error Type Analysis. Table 10 reports the distribution of error types among invalid instances. The most frequent error type was incorrect reference resolution, accounting for 34.7% of invalid cases. This error typically occurred when a query contained pronouns or demonstratives whose antecedents were ambiguous across multiple previous turns. The second most frequent error type was missing or weakly grounded arguments, accounting for 26.6% of invalid cases. These cases involved API arguments that could not be fully inferred from the dialogue context. Wrong API selection and hallucinated argument values were less frequent, accounting for 14.5% and 11.3% of invalid cases, respectively.

To evaluate whether models trained on the Mobile Dialogue Multi-Turn dataset generalize beyond the original synthetic benchmark, we constructed a 1,000-example human-written realistic validation set. Three developers familiar with mobile assistant systems collaboratively drafted, revised, and annotated user-style dialogue instances spanning all 41 APIs. The instances were designed to mimic common properties of mobile user utterances, including ASR-style noise, colloquial phrasing, pronoun/deixis, ellipsis, and self-corrections.

The construction process was guided by established LLM-augmentation and data-augmentation pipelines (Gu and Yang, 2024; Zhou et al., 2024; Li et al., 2025). The three developers divided the initial annotation work while following the same annotation guidelines. Annotators were instructed not to merely paraphrase the original synthetic queries, but to introduce user-like phenomena commonly observed in mobile assistant interactions, such as incomplete requests, informal wording, speech-recognition-like errors, referential expressions, and mid-utterance self-corrections. At the same time, they were required to preserve the original API intent and required arguments.

Each instance was checked for API alignment, reference resolution, and argument groundability. Instances with changed intent, ambiguous references, unsupported arguments, or unnatural rewrites were revised or removed. To improve reliability, a subset of instances from each developer was cross-checked by another developer. Disagreements were discussed among the developers and resolved by consensus. After revision and quality filtering, the final set contained 1,000 examples. This set was constructed separately from the training split and was used only for evaluation, not for training or model selection.

C.4 Final Dataset Statistics

Main Benchmark. The final filtered dataset comprises 17,453 multi-turn dialogue instances after excluding unresolved cases. We summarize the dataset distribution by current turn and reference distance below.

- **Current Turn Distribution**
  - Turn 2: 2,023 instances (11.6%)
  - Turn 3: 3,852 instances (22.1%)
  - Turn 4: 4,898 instances (28.1%)
  - Turn 5: 5,246 instances (30.1%)
  - Turn 6: 1,434 instances (8.2%)
- **Reference Distance Distribution**
  - Distance 1: 8,822 instances (50.5%) – previous turn
  - Distance 2: 4,873 instances (27.9%) – two turns back
  - Distance 3: 2,568 instances (14.7%) – three turns back
  - Distance 4: 980 instances (5.6%) – four turns back

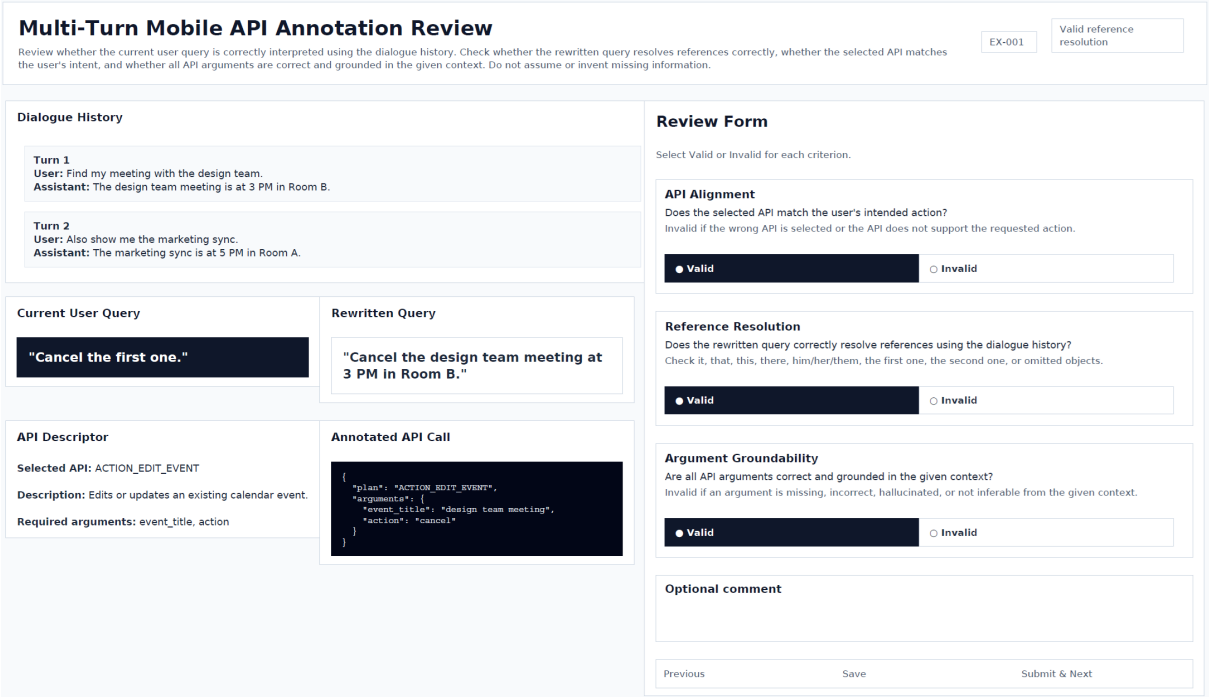


Figure 4: Web-based annotation interface for human evaluation.

  - Distance 5: 210 instances (1.2%) – five turns back

Reference distance is defined as the difference between the current turn and the turn containing the antecedent required to resolve the current query. Overall, 49.5% of the instances refer to information beyond the immediately preceding turn, showing that the benchmark includes substantial long-range contextual dependencies.

Realistic Validation Set. The realistic validation set contains 1,000 human-written multi-turn dialogue instances spanning all 41 APIs. It was constructed separately from the training data and used only for evaluation. We summarize its distribution by current turn and reference distance below.

- **Current Turn Distribution**
  - Turn 2: 103 instances (10.3%)
  - Turn 3: 180 instances (18.0%)
  - Turn 4: 252 instances (25.2%)
  - Turn 5: 284 instances (28.4%)
  - Turn 6: 181 instances (18.1%)
- **Reference Distance Distribution**
  - Distance 1: 395 instances (39.5%) – previous turn
  - Distance 2: 292 instances (29.2%) – two turns back
  - Distance 3: 186 instances (18.6%) – three turns back
  - Distance 4: 101 instances (10.1%) – four turns back
  - Distance 5: 26 instances (2.6%) – five turns back

Reference distance is defined as the difference between the current turn and the turn containing the antecedent required to resolve the current query. Overall, 60.5% of the realistic validation instances require resolving references beyond the immediately preceding turn.

C.5 Examples of Filtered Data

Representative examples of filtered data include:

Example 1: Filtered due to missing the required parameter ('to' recipient) for the send_email API.

- **Previous Conversation**: Could you build a fresh XML file named data.xml? → Your new XML file 'data.xml' is ready. Access it at: content://documents/data.xml
- **Query**: Share that file with the team by email
- **Rewritten Query**: Share content://documents/data.xml with the team by email

Parameter	Value
Models	microsoft/Phi-4-mini-instruct Qwen/Qwen3-4B Qwen/Qwen3-1.7B Qwen/Qwen3-0.6B meta-llama/Llama-3.2-3B-Instruct HuggingFaceTB/SmolLM3-3B zai-org/glm-edge-1.5b-chat zai-org/glm-edge-4b-chat ibm-granite/granite-3.3-2b-instruct
Training epochs	4
Batch size per device	2
Gradient accumulation steps	4
Learning rate	5×10^{-5}
Precision	bf16
Gradient checkpointing	Enabled
Max gradient norm	1.0
LoRA rank (r)	16
LoRA alpha (α)	32
LoRA dropout rate	0.05
LoRA target modules	all-linear
Optimizer	AdamW
Save strategy	epoch
Save total limit	8
Logging steps	50

Table 11: Detailed Training Setup and Hyperparameters

Example 2: Filtered due to incorrect referencing when no referent existed.

- **Previous Conversation**: Open the music app and start playing some tunes. → Your music app is open and tunes are playing now.
- **Query**: What songs are in this playlist?
- **Rewritten Query**: What songs are in the *Your music app is open and tunes are playing now* playlist?

Example 3: Filtered due to inability to generate a query for navigate_to_location API, as the conversation history lacked location information.

- **Previous Conversation**: Update content://contacts/people/1011 name to Samantha Smith. → Name changed to Samantha Smith. Here's your updated contact: content://contacts/people/1011
- **Query**: Navigate to her location.
- **Rewritten Query**: Navigate to Samantha Smith's location.

D Training Setup Details

D.1 Model and Hyperparameter Configuration

We trained all models using the hyperparameters specified in Table 11. For LoRA fine-tuning, we targeted all linear modules. The corresponding trainable parameter ratios for each model are reported in Table 12.

We selected the final hyperparameters based on validation macro-accuracy on the development set. Specifically, we explored a small set of commonly used LoRA configurations, including learning rates in $\{1 \times 10^{-5}, 3 \times 10^{-5}, 5 \times 10^{-5}\}$, LoRA ranks in $\{8, 16\}$, LoRA α values in $\{16, 32\}$, and dropout values in $\{0.05, 0.1\}$. The final setting used a learning rate of 5×10^{-5}, LoRA rank 16, LoRA α of 32, and dropout of 0.05.

We employed the AdamW optimizer and bf16 precision for computational efficiency. Training was performed on NVIDIA A100 80GB GPUs. Across all training runs, preliminary experiments, and evaluation-related runs, the total computational budget was approximately 100 GPU-hours on NVIDIA A100 80GB GPUs. A single main training run for each model took about 1 GPU-hour.

D.2 Target Formats for Compared Methods

Table 13 summarizes the input and target-output formats used for each compared method. All methods use the same instruction-style serialization unless otherwise specified. The main difference lies in the supervised target sequence.

```
You are a helpful assistant capable of selecting appropriate tools based on user queries and generating corresponding parameters.
Use information from the conversation history when relevant. Only use parameter values that are explicitly stated or can be reasonably inferred from the query.
If no tool matches the query, set the tool to 'None'.

Available tools:
{tools}

Conversation History: {conversation_history}
User Query: {query}

{"plan":"{plan}","arguments":{arguments}}
```

Listing 7: SFT prompt template for the Baseline.

READ uses the same input serialization as Listing 7, but modifies the response instruction and target output so that the rewritten query is generated before the API name and arguments.

```
<same prompt prefix as the Baseline>
Return compact JSON only with keys "rewritten_query", "plan", and "arguments". Always include all three keys.
```

Model Name	Parameters	Target Modules	Trainable Parameters (%)
microsoft/Phi-4-mini-instruct	3.8B	all linear layers	0.81
Qwen/Qwen3-4B	4B	all linear layers	0.81
Qwen/Qwen3-0.6B	0.75B	all linear layers	1.33
Qwen/Qwen3-1.7B	2.03B	all linear layers	0.85
meta-llama/Llama-3.2-3B-Instruct	3.21B	all linear layers	0.75
HuggingFaceTB/SmolLM3-3B	3.08B	all linear layers	0.97
zai-org/glm-edge-1.5b-chat	1.59B	all linear layers	1.00
zai-org/glm-edge-4b-chat	4.33B	all linear layers	0.74
ibm-granite/granite-3.3-2b-instruct	2.53B	all linear layers	1.10

Table 12: Summary of Models, Parameter Counts, Targeted Modules for Model Training, and Corresponding Ratios of Trainable Parameters

Method	Input	Target Output
Baseline	History + Query + API Schema	Plan + Args
Baseline + Resampling	History + Query + API Schema	Plan + Args
Baseline + Retrieval	Retrieved Turns + Query + API Schema	Plan + Args
Two-stage QR (Stage 1)	History + Query	Rewrite
Two-stage QR (Stage 2)	Rewrite + API Schema	Plan + Args
READ	History + Query + API Schema	Rewrite + Plan + Args
READ + Resampling	History + Query + API Schema	Rewrite + Plan + Args
Reverse-order	History + Query + API Schema	Plan + Args + Rewrite
Oracle Rewrite	Gold Rewrite + API Schema	Plan + Args

Table 13: Input and target-output formats for the compared methods and ablations.

```
The value of "arguments" must always be an object.

{"rewritten_query":"{rewritten_query}","plan":"{plan}","arguments":{arguments}}
```

Listing 8: Changed response instruction and target output for READ.

The reverse-order ablation uses the same input serialization but places the API prediction before the rewritten query.

```
<same prompt prefix as the Baseline>
Return compact JSON only with keys "plan", "arguments", and "rewritten_query".
Always include all three keys. The value of "arguments" must always be an object.

{"plan":"{plan}","arguments":{arguments},"rewritten_query":"{rewritten_query}"}
```

Listing 9: Changed response instruction and target output for the reverse-order ablation.

For two-stage QR baseline, we first train a rewrite-only model using the same gold rewrite supervision, and then feed the generated rewrite into the same planner used for the Oracle Rewrite setting.

For Oracle Rewrite, the gold rewritten query replaces the original dialogue history and current query as the model input, and the target output contains only the API name and arguments. This setting measures the upper-bound performance when context resolution is assumed to be correct. The resampling variants use the same SFT target formats as their corresponding base methods. The difference is applied only during inference, where invalid outputs are regenerated under the same schema constraints.

D.3 Retrieval Baseline Configuration

For the retrieval-augmented baseline, we encode the current user query and each previous dialogue turn using a dense retrieval encoder. We use `BAAI/bge-small-en-v1.5` (Xiao et al., 2024), a lightweight BERT-style embedding model with approximately 33M parameters. Each history turn is ranked by the cosine similarity between its embedding and the current-query embedding, with a small recency bonus added to prefer more recent turns when relevance scores are similar. The top-ranked turns are then inserted into the planner prompt in place of the full dialogue history.

We compare $k \in \{1, 2, 3\}$ and report the best-performing retrieval setting, $k = 2$, in the main experiments, giving the retrieval baseline a favorable top-k configuration among the tested retrieval sizes. All other planner, prompt-format, and target-output settings are kept identical to the Schema-aware Planner baseline. Therefore, the comparison isolates the effect of replacing full-history conditioning with retrieval-based history selection.

E Failure and Success Case Analysis

We analyzed success and failure cases of the Baseline and READ. In particular, we examine both the Baseline errors corrected by READ and the residual errors that remain challenging even after explicit decomposition.

In the main results, READ improves macro-averaged accuracy from 70.5% to 74.7% (+4.2 points). In this case-level analysis, we analyze pooled model-instance predictions across the six target models on Turns 2–6 under the non-resampled setting. READ corrects 2,902 Baseline errors, while changing 1,455 Baseline-correct predictions into incorrect predictions, resulting in a net gain of 1,447 additional correct predictions. At the same time, 7,221 model-instance predictions remain incorrect (27.2%).

E.1 Error Taxonomy

Each failed prediction is assigned one primary error label. These labels are not intended to represent mutually exclusive linguistic phenomena, but rather to capture the dominant failure mode. When multiple issues are present, we assign the primary label based on the earliest stage in the READ output sequence that affects the final executable call. Specifically, rewrite-level errors are labeled before API-selection or argument-value errors, and argument errors are assigned only when both the rewrite and API selection are correct.

Primary error labels are assigned using a rule-based script that compares the generated rewrite, API name, and arguments against the gold annotation. This taxonomy is used for diagnostic analysis of READ's residual errors, rather than as a separate human-annotated benchmark.

We use six error categories: incorrect argument value after correct rewrite, semantic rewrite drift, reference/entity grounding error in rewrite, surface-form mismatch under exact match, incorrect API selection, and formatting/schema error. Reference/entity grounding error includes cases where the rewrite selects an incorrect antecedent, omits a slot-bearing entity from the history, or adds unsupported entity details not grounded in the dialogue. Surface-form mismatch refers to cases that are semantically close to the gold output but fail under strict exact match due to formatting or canonical string differences.

Table 14 summarizes the residual error taxonomy. The largest categories are incorrect argument value after correct rewrite (2,542 cases), semantic rewrite drift (2,393 cases), and reference/entity grounding error in rewrite (1,236 cases). The top two categories alone account for 68.3% of all residual errors, while reference/entity grounding errors in the rewrite account for 17.1%. Formatting errors are negligible.

This taxonomy suggests that, after rewriting, coarse API selection is no longer the dominant residual error source. Instead, most remaining errors are related to fine-grained argument grounding or semantic shifts in the rewritten query. In particular, semantic rewrite drift remains a major error source, showing that explicit rewriting can also introduce subtle meaning changes before planning. In other words, rewriting substantially helps contextualization, but accurate grounding and semantic faithfulness remain important residual challenges.

E.2 Baseline Failure Cases Corrected by READ

The first set of examples is organized around four recurring benefits of rewriting: reference disambiguation, slot completion and attachment recovery, API disambiguation, and search/navigation grounding. These examples show that READ is especially useful when the user query is deictic or elliptical, and when the model must recover an older referent, carry over a slot-bearing identifier, or convert an underspecified request into an executable form.

The purpose of this section is not merely to list improved examples, but to illustrate the mechanism behind the performance gains. In the strongest cases, READ resolves ambiguous pronouns to the correct contact or event, recovers missing URIs or attachments, and makes the intended action explicit enough for the planner to produce the correct API call.

Example 1: Reference Disambiguation. Context. The recent-call list contains multiple candidates, including Mia, Noah, and Olivia. The previous assistant response identifies Olivia as the missed caller, while Mia only appears as an older call-log entry. Therefore, the follow-up request must resolve the pronoun to the correct entry in the call history.

User query. Dial her

Gold target. API = `dial`;
phone number = `+44-20-7946-0999`

Baseline prediction. API = `dial`;
phone number = `+1-213-555-0156`

Category	Residual Failures	Share
Incorrect argument value after correct rewrite	2542	35.2%
Semantic rewrite drift	2393	33.1%
Reference/entity grounding error in rewrite	1236	17.1%
Surface-form mismatch under exact match	554	7.7%
Incorrect API selection	484	6.7%
Formatting/schema error	12	0.2%

Table 14: Residual error taxonomy for READ under the non-resampled setting.

READ prediction. Rewrite = "Dial Olivia at +44-20-7946-0999"; API = `dial`; phone number = `+44-20-7946-0999`.

Interpretation. The rewrite recovers the correct antecedent and explicitly exposes the correct phone number. This shows that READ can improve reference resolution even when the surface form of the query is highly underspecified.

Example 2: Slot Completion and Attachment Recovery. Context. The user first captures a photo and then asks to send it by email. To execute this request, the selected image URI must be carried over as the attachment of the email call.

User query. Email it to `boss@company.com`

Gold target. API = `send_email`; recipient = `boss@company.com`; attachment = `content://temp/IMG_20240605_153210.jpg`

Baseline prediction. API = `send_email`; recipient = `boss@company.com`; attachment = missing

READ prediction. Rewrite = "Email content://temp/IMG_20240605_153210.jpg to boss@company.com"; API = `send_email`; recipient = `boss@company.com`; attachment = `content://temp/IMG_20240605_153210.jpg`.

Interpretation. The rewrite preserves the slot-bearing file URI, allowing the planner to output complete email arguments rather than an underspecified action containing only the recipient.

Example 3: API Disambiguation. Context. A contact URI is first exposed in the dialogue, and an unrelated email address is mentioned later. The final request asks to open the original referent.

User query. Open it

Gold target. API = `ACTION_VIEW_CONTACT`; contact URI = `content://contacts/people/1024`

Baseline prediction. API = `ACTION_OPEN_CONTENT`; URI = `content://contacts/people/1024`

READ prediction. Rewrite = "Open content://contacts/people/1024"; API = `ACTION_VIEW_CONTACT`; contact URI = `content://contacts/people/1024`.

Interpretation. The Baseline recovers the correct URI but maps it to a generic content-opening action. In contrast, READ better preserves the object type associated with the grounded URI and selects the contact-specific API. This suggests that READ can help not only with recovering entity strings, but also with disambiguating which API action should be associated with the entity.

Example 4: Search and Navigation Grounding. Context. An earlier event is located at the Admin Office, while a nearby coffee shop is later mentioned for a different reminder. The final request should navigate to the event location, not to the most recently mentioned place.

User query. Navigate there.

Gold target. API = `ACTION_NAVIGATE_TO_LOCATION`; location = `Admin Office`

Baseline prediction. API = `ACTION_NAVIGATE_TO_LOCATION`; location = `Central Perk Café, 200 Main St, Springfield, IL 62705`

READ prediction. Rewrite = "Navigate to Admin Office."; API = `ACTION_NAVIGATE_TO_LOCATION`; location = `Admin Office`.

Interpretation. The rewrite resolves the deictic destination to the intended earlier location rather than the most recent distractor. This illustrates the value of rewriting as interaction depth increases.

E.3 Residual Cases That Remain Difficult for READ

The corrected cases above show where rewriting is most directly helpful. To complement them, we also include a small residual contrast set where difficulties remain even after rewriting. The following examples focus on the two largest residual categories in Section E: incorrect argument value after correct rewrite and semantic rewrite drift, as well as one reference/entity grounding failure.

Example 5: Residual Argument Grounding Error. Context. The dialogue establishes only that the requested file should be a Word document. The rewrite is semantically correct, but the final API call must still select the exact document MIME type.

> **User query.** Create a new one named `Draft.docx`.
>
> **Gold target.**
> API = `ACTION_CREATE_DOCUMENT`;
> MIME type = `application/vnd.openxmlformats-officedocument.wordprocessingml.document`;
> initial name = `Draft.docx`
>
> **READ prediction.**
> Rewrite = "Create a new Word document named Draft.docx.";
> API = `ACTION_CREATE_DOCUMENT`;
> MIME type = `application/msword`; initial name = `Draft.docx`.

Interpretation. This example shows that even when READ correctly resolves the user intent, the downstream generator can still miss the exact slot value required for execution. The model selects the legacy Word MIME type rather than the DOCX-specific MIME type required by the file extension.

Example 6: Residual Semantic Rewrite Drift. Context. In the previous turn, the Spotify playlist `Good Morning` is started. The follow-up request should refer to the playlist content, not to the application itself.

> **User query.** Um, yeah, can you show me all the songs in it?
>
> **Gold target.** API = `ACTION_VIEW_PLAYLIST`; playlist name = `Good Morning`
>
> **READ prediction.** Rewrite = "Um, yeah, can you show me all the songs in Spotify?";
> API = `ACTION_VIEW_PLAYLIST`;
> playlist name = `Spotify`.

Interpretation. Here, the rewrite preserves the naturalness of the interaction but shifts away from the intended object. It replaces the target playlist with the Spotify app, thereby changing the meaning of the request. This case shows that although explicit rewriting improves context resolution overall, semantic faithfulness of the intermediate representation remains important.

Example 7: Residual Reference/Entity Grounding Failure. Context. The previous turn extracts an email address from a contact URI. The follow-up request should reopen the same contact object, rather than treating the extracted email string itself as the contact.

Turn	Improvement over Baseline
Turn 2	+1.29
Turn 3	+3.04
Turn 4	+4.10
Turn 5	+5.14
Turn 6	+7.09

Table 15: Turn-wise improvement of READ over the Baseline.

> **User query.** Display that contact
>
> **Gold target.** API = `ACTION_VIEW_CONTACT`;
> contact URI = `content://contacts/people/721`
>
> **READ prediction.** Rewrite = "Display contact samantha.lee@contacts.net";
> API = `ACTION_VIEW_CONTACT`;
> contact URI = `samantha.lee@contacts.net`.

Interpretation. This failure case shows that READ can collapse an entity mention into one of its attributes. As a result, the rewrite appears plausible on the surface but grounds the reference incorrectly.

E.4 Turn-wise Gains and Corrected Baseline Errors

READ's accuracy advantage increases as turn depth grows. Consistent with the turn-wise macro-average results in the main text, the improvement is only +1.29 points at Turn 2, but increases to +7.09 points at Turn 6. Table 15 summarizes these turn-wise gains. This pattern aligns with the main motivation for rewriting: as the relevant referent becomes farther from the current query, explicit reformulation becomes more valuable.

From a second perspective, we examine all cases where the Baseline prediction is incorrect but READ is correct. This analysis is based on pooled model-instance predictions and includes 2,902 corrected Baseline errors. Table 16 summarizes the types of Baseline errors corrected by READ. This number is larger than the net gain of 1,447 because the net gain subtracts the 1,455 Baseline-correct cases that READ changes into incorrect predictions.

The dominant pattern among corrected cases is that the Baseline already selects the correct API but fails to recover the correct argument values from the dialogue context. This type accounts for 88.35% of all corrected cases. This supports the view that READ's main benefit lies not in generic action selection, but in improved reference resolution and slot grounding.

Corrected Baseline Error	Count	Share
Correct API, incorrect arguments	2564	88.35%
Incorrect API and incorrect arguments	302	10.41%
Incorrect API, correct arguments	36	1.24%

Table 16: Types of Baseline errors corrected by READ.

Overall, the corrected-error analysis suggests that READ's main benefit is improved fine-grained argument grounding rather than merely changing coarse API choices. At the same time, the remaining failures in later turns indicate that long-range entity grounding and semantic faithfulness in rewriting remain important open challenges.

F On-Device Evaluation Details

We measured on-device latency with our Android evaluator, a modified version of the open-source SmolChat-Android project using `llama.cpp` as the local inference backend. All models were converted to `GGUF` format using `llama.cpp` and quantized with the `Q4_K_M` 4-bit quantization scheme. For each model, Baseline and READ used the same quantized model file.

The evaluator separately measures prompt prefill time and autoregressive decode time, and total latency is defined as their sum. During batch evaluation, model weights remain loaded in memory, but the conversation state, prompt context, and `llama.cpp` KV cache are reset before each example. No persistent prompt or session cache is reused across examples.

Baseline and READ were evaluated under the same app-side runtime configuration, including context size, thread count, memory-mapping options, and evaluator-side generation settings, except for differences required by their target output formats. Latency values in Table 4 are normalized within each model family using the smallest Baseline model as the reference.