ACR: Adaptive Context Refactoring via Context Refactoring Operators for Multi-Turn Dialogue

Jiawei Shen[1*], **Jia Zhu**[1*†], **Hanghui Guo**[2], **Weijie Shi**[3], **Yue Cui**[4], **Qingyu Niu**[1], **Guoqing Ma**[1], **Yidan Liang**[1], **Jingjiang Liu**[1], **Yilin Wang**[1], **Shimin Di**[2], **Jiajie Xu**[5†]

[1]Zhejiang Key Laboratory of Intelligent Education Technology and Application, Zhejiang Normal University
[2]School of Computer Science and Engineering, Southeast University
[3]Department of Computer Science and Engineering, Hong Kong University of Science and Technology
[4]Alibaba Group, [5]School of Computer Science and Technology, Soochow University

Abstract

Large Language Models (LLMs) have shown remarkable performance in multi-turn dialogue. However, in multi-turn dialogue, models still struggle to stay aligned with what has been established earlier, follow dependencies across many turns, and avoid drifting into incorrect facts as the interaction grows longer. Existing approaches primarily focus on extending the context window, introducing external memory, or applying context compression, yet these methods still face limitations such as **contextual inertia** and **state drift**. To address these challenges, we propose the **A**daptive **C**ontext **R**efactoring **(ACR)** Framework, which dynamically monitors and reshapes the interaction history to mitigate contextual inertia and state drift actively. ACR is built on a library of context refactoring operators and a teacher-guided self-evolving training paradigm that learns when to intervene and how to refactor, thereby decoupling context management from the reasoning process. Extensive experiments on multi-turn dialogue demonstrate that our method significantly outperforms existing baselines while reducing token consumption. Our code is available at https://github.com/ClannadKno/multi-turn.

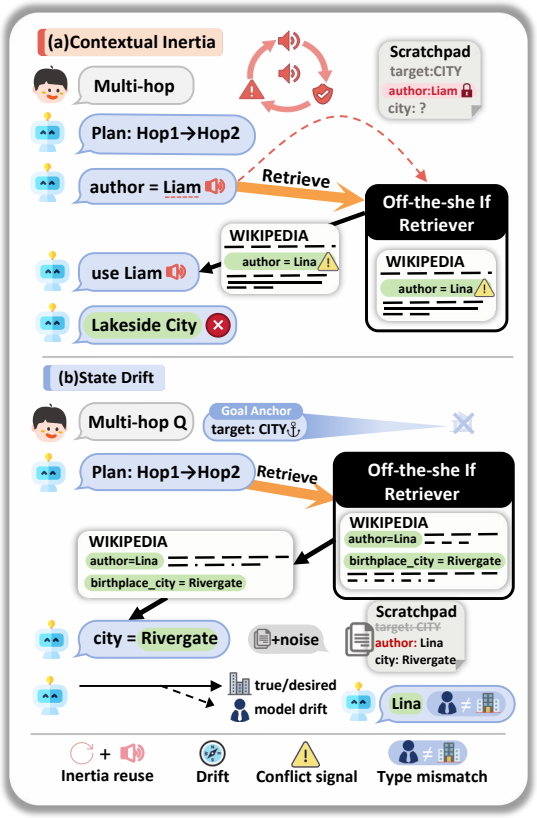

Figure 1: Illustration of two Challenges in multi-turn dialogue: *(a) contextual inertia* and *(b) state drift*.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in language understanding and generation within multi-turn dialogue scenarios (Ouyang et al., 2022; Zhu et al., 2025b; Zhang et al., 2025). However, as interaction turns increase, maintaining contextual consistency, modeling long-range dependencies, and ensuring factual faithfulness remain prohibitive challenges (Dziri et al., 2022; Liu et al., 2024).

To address these challenges, early studies attempted to enhance the model's ability to handle long-range dialogues by expanding the context window (Chen et al., 2024; Dao et al., 2022; Cuconasu et al., 2024). These approaches allow the model to focus on more complete conversation histories, thereby partially alleviating the coherence issues caused by information loss. However, such methods primarily increase the visible information capacity without addressing redundancy in the context. This redundant or even erroneous historical content may be included, potentially introducing noise that hinders subsequent reasoning.

In this context, existing studies attempt to fun-

*These authors contributed equally.
†Corresponding author.
e-mail: 1185096117@zjnu.edu.cn, jiazhu@zjnu.edu.cn, xujj@suda.edu.cn

Findings of the Association for Computational Linguistics: ACL 2026, pages 3149–3167
July 2-7, 2026

damentally address the challenges in multi-turn dialogue by introducing external information and applying context compression. The former seeks to extract critical details from implicit contextual dependencies into explicit, controllable forms. For instance, external memory mechanisms structure and store key information, enabling the model to quickly access and update these details during multi-turn dialogue without relying on lengthy historical context. This helps to avoid information loss or interference caused by excessively long context, thereby maintaining coherence and accuracy in multi-turn dialogue (Bae et al., 2022; Zhong et al., 2024). Meanwhile, Retrieval-Augmented Generation (RAG) queries external knowledge sources to supply explicit evidence, reducing the hallucinations and inconsistencies often caused by volatile memory or missing facts (Lewis et al., 2020; Jiang et al., 2022; Su et al., 2024; Guo et al., 2025b; Zhu et al., 2025a). Conversely, context compression strategies reduce lengthy interactions into shorter, information-dense representations, improving the model's accessibility to key information in subsequent reasoning steps (Jiang et al., 2023; Chuang et al., 2024). Nevertheless, as shown in Figure 1, current methods still face challenges:

Challenge 1: Contextual Inertia. Although approaches such as context window extension, external memory, and RAG enrich history and evidence, these methods primarily focus on information provision. They lack adequate coupling with reasoning error-correction mechanisms during generation. In multi-turn dialogue, the model tends to develop path dependency on the existing context, where minor logical deviations or factual errors in earlier turns are continuously incorporated into subsequent reasoning. Since the model is inclined to maintain narrative coherence, errors are not only hard to expose in time but may also be gradually reinforced. Consequently, the model may fall into a locally consistent but globally erroneous cognitive loop, weakening its ability to trace key evidence and revise assumptions.

Challenge 2: State Drift. Existing methods typically represent historical dialogues as unstructured flat sequences. Even with external memory, retrieval, or compression techniques, they still rely on fixed storage and recall strategies, lacking explicit anchoring mechanisms for state evolution. As turns accumulate, initial global constraints and intermediate goals may become diluted by local questions and noise. While these pieces of information may not be significant at the moment, they may become critical at later reasoning nodes. Once this key intermediate information is submerged by noise or lost during compression, subsequent reasoning proceeds based on incomplete state representations. This causes the model to drift from updated constraints, ultimately resulting in logical breaks and failed objectives.

To address the above challenges, we propose an **A**daptive **C**ontext **R**efactoring **(ACR)** framework, which proactively manages and refactors the interaction history when needed to improve the stability and reliability of multi-turn dialogue. Specifically, to tackle contextual inertia and state drift, we first construct a library of context refactoring operators covering six strategy types. Building on this library, we introduce a **teacher-guided self-evolving** training paradigm that enables the model to learn when to refactor and how to select and execute refactoring strategies. This paradigm iteratively optimizes the router and the refactorer in a closed loop, resulting in an LLM with monitoring capabilities that continuously diagnose the evolving history context. When signs of drift or inertia are detected, the router selects an appropriate operator to trigger intervention. The refactorer then applies the corresponding strategy to produce a refactored context, which replaces the original history and is fed into the reasoning model for subsequent inference.

Our contributions are summarized as follows:

- We propose ACR, which dynamically monitors and reshapes the interaction history to actively mitigate the challenges of contextual inertia and state drift in multi-turn dialogue.
- We introduce a library of **context refactoring operators** and a **Teacher-Guided Self-Evolving** training paradigm. This method decouples context management from reasoning, enabling the LLM to internalize refactoring capabilities without expensive reinforcement learning.
- Extensive experiments on long-context tasks demonstrate that our method significantly outperforms existing baselines while reducing token consumption. The results validate that context refactoring is a more efficient path to long-horizon reasoning than enhancing logic via RL.

2 Related Work

Large language models (LLMs) have exhibited remarkable capabilities in multi-turn dialogue (Liu

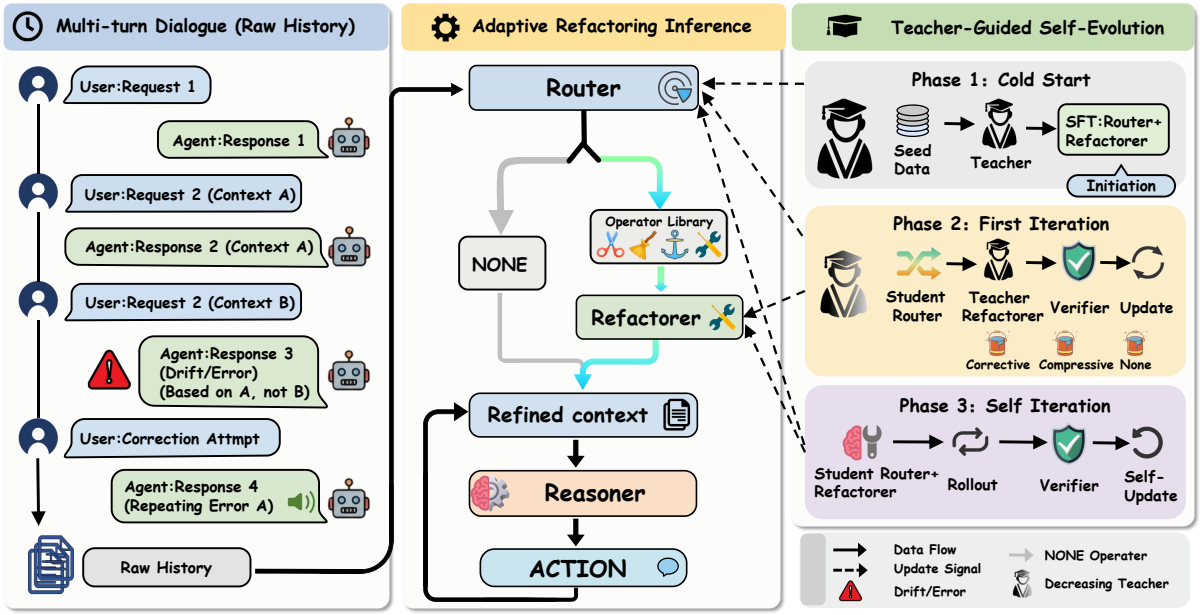

Figure 2: The framework of ACR. The pipeline includes Semantic Routing, where the Router selects the appropriate context operator, and Adaptive Refactoring, which refines the context for improved reasoning. The framework evolves through a Teacher-Guided Self-Evolution training, transitioning from supervised learning to autonomous decision-making, enhancing reasoning accuracy and efficiency.

et al., 2024; Bai et al., 2024). However, their performance often degrades as the number of interaction turns increases, primarily due to difficulties in maintaining contextual consistency, capturing long-range dependencies, and ensuring factual accuracy.

To overcome these issues, researchers have proposed a variety of strategies. In early studies, methods like DRAGIN (Su et al., 2024), DioR (Guo et al., 2025b), and SEAKR (Yao et al., 2025) dynamically trigger retrieval based on uncertainty metrics or classifier-based detection. While effective at supplying explicit evidence, these approaches suffer from **Contextual Inertia**. They primarily focus on information provision rather than reasoning correction; if the historical context already contains logical deviations, retrieving additional evidence based on flawed premises often reinforces rather than resolves the error loop.

To handle the problem of long horizons, HippoRAG (Jimenez Gutierrez et al., 2024) and ITER-RETGEN (Shao et al., 2023) utilize knowledge graphs or iterative retrieval, while RECOMP (Xu et al., 2024) employs selective compression to save tokens. However, these methods typically treat history as a static accumulation or reduce it via indiscriminate summarization. Lacking explicit anchoring mechanisms, they are vulnerable to **State Drift**, where critical global constraints are submerged by local noise or lost during compression, leading to reasoning on incomplete state representations.

Reinforcement Learning methods optimize agent policies for multi-step reasoning. Search-R1 (Jin et al., 2025) treats search as an environment with token masking, and StepSearch (Zheng et al., 2025) achieves process-level supervision for LLM search via step-wise Proximal Policy Optimization incorporating information gain and redundancy penalties, thereby significantly improving reasoning performance on complex multi-hop QA tasks. They lack the ability to actively refactor the context, making them insufficient to break the path dependency inherent in long-context reasoning.

3 Preliminary

Multi-turn dialogue. Formally, we define a multi-turn dialogue session as a sequential process $\mathcal{S} = \{(u_1, a_1), \ldots, (u_T, a_T)\}$, where u_t denotes the user instruction at turn t, and a_t represents the corresponding agent response. At any time step t, the History Context available to the model consists of the accumulated sequence of previous interactions:

$$H_t = [u_1, a_1, \ldots, u_{t-1}, a_{t-1}]. \quad (1)$$

Standard Reasoning Paradigm. Given an LLM parameterized by θ, the standard objective is to generate the response a_t by maximizing the con-

ditional likelihood given the raw history and the current instruction:

$$a_t = \underset{a \in \mathcal{V}^*}{\operatorname{argmax}} P_\theta(a \mid H_t, u_t). \tag{2}$$

In this paradigm, the model relies solely on the implicit attention mechanism to extract relevant features from the potentially lengthy and noisy H_t.

Objective. Our goal is to transcend the limitations of conditioning on raw history. We aim to identify an optimized context representation $\tilde{H}_t$ derived from H_t, such that the likelihood of generating the optimal response a_t^* is maximized by $\max P_\theta(a_t^* \mid \tilde{H}_t, u_t)$.

In the following sections, we introduce a mechanism to explicitly construct $\tilde{H}_t$ via a library of context refactoring operators.

4 Proposed Framework: ACR

In this section, we present **A**daptive **C**ontext **R**efactoring **(ACR)**, an innovative multi-turn dialogue framework, as illustrated in Figure 2. ACR monitors the evolving history context during inference and refactors it on demand to improve the stability and reliability of multi-turn dialogue.

4.1 Adaptive Context Refactoring Inference

Adaptive context refactoring inference is a core component of the ACR framework, which introduces an external controller model to supervise and refactor the dialogue history on demand. Unlike prior methods that passively concatenate and accumulate dialogue history, ACR preserves the full history H while introducing an external refactoring controller to dynamically supervise the reasoning process. When signals are detected, the controller refactors the history in a *need-driven* and *structured* manner, providing the downstream reasoner with a higher signal-to-noise input. ACR manages the reasoning context via two stages:

Semantic Routing. We instantiate a semantic router that continuously encodes the current history H and predicts a probability distribution over refactoring operators defined in Section 4.2. The router identifies the contextual status of the current turn: whether the history should be summarized, corrected, or left unchanged by selecting the NONE operator. This stage shifts context handling from indiscriminate full-history feeding to problem-aware strategy selection.

Dynamic Refactoring. When the router selects a non-NONE operator, the system *hot-swaps* the corresponding Refactoring LoRA, which transforms the raw history H into a structured and low-noise refactored context $\tilde{H}$ according to the chosen operator. We then replace the original history with $\tilde{H}$ as the single, coherent information source for the downstream reasoning model.

This closed-loop design reduces the attention and retrieval burden over long histories and suppresses the repeated reuse and rationalization of early mistakes, thereby improving the stability and controllability of multi-turn dialogue.

4.2 The Library of Context Refactoring Operators

In multi-turn dialogue, the raw interaction history H is often laden with redundancy, stochastic noise, hallucinations, or logical deadlocks. Directly conditioning an LLM on such raw H exacerbates state drift. To address this, we define a comprehensive library of **Context Refactoring Operators**, denoted as $\mathcal{A}$. These operators are not merely heuristic truncation rules but are grounded in information theory, aiming to transform linear, flat logs into structured, high-semantic-density memory representations. We categorize the six operators into three distinct functional groups: Information Density Optimization, Logical Flow Control, and Attention Management. Additionally, an **identity operator** is included to handle optimal contexts.

4.2.1 Category 1: Information Density Optimization

State Abstraction. In multi-turn dialogue, intermediate steps often contain redundant exploration that no longer influences future decisions. Relying on the Markov Assumption, $\mathcal{O}_{\text{abs}}$ compresses the serialized action-observation history into a semantic State Snapshot, while retaining the most recent user instruction to ensure task continuity.

$$\mathcal{O}_{\text{abs}}(H) = \mathcal{S}_\phi(H_{<t}) \oplus x_t, \tag{3}$$

where $\mathcal{S}_\phi$ denotes the state summarization function, x_t is the current query, and $\oplus$ represents concatenation.

Noise Filtering. Retrieved contexts often contain distractor documents that are orthogonal to the current intent. $\mathcal{O}_{\text{filter}}$ performs surgical text pruning by evaluating the semantic relevance of each text unit u_i within H, filtering out segments that fall below a relevance threshold τ:

$$\mathcal{O}_{\text{filter}}(H, x_t) = u_i, \tag{4}$$

where $u_i \in \{H \mid \text{Sim}(u_i, x_t) > \tau\}$. This operator maximizes the signal-to-noise ratio within the limited context window.

4.2.2 Category 2: Logical Flow Control

Fact Rectification. Long-context generation is prone to hallucinations and the persistence of obsolete or incorrect claims. Unlike standard approaches that append corrections to the end, $\mathcal{O}_{\text{rect}}$ utilizes an external verifier $\mathcal{V}$ to identify hallucinated propositions and performs in-place editing. Specifically, an external verifier $\mathcal{V}$ checks each context unit u_i and decides whether it is factually valid. If u_i is verified as true, it is kept unchanged; otherwise, it is rewritten by a rewriting function $\mathcal{R}$.

$$\begin{aligned} &\mathcal{O}_{\text{rect}}(H) = [\tilde{u}_1, \ldots, \tilde{u}_n], \\ &\text{where } H = [u_1, \ldots, u_n], \end{aligned} \tag{5}$$

$$\tilde{u}_i = \begin{cases} u_i, & \text{if } \mathcal{V}(u_i) = \texttt{true}, \\ \mathcal{R}(u_i), & \text{otherwise.} \end{cases} \tag{6}$$

Path Pruning. Complex reasoning often leads to logical loops or incorrect exploration branches. Analogous to backtracking in search algorithms, $\mathcal{O}_{\text{prune}}$ identifies the point of logical bifurcation or failure, denoted as k. It explicitly truncates the history after k, rolling the context back to the nearest clean state to prevent error cascading:

$$\mathcal{O}_{\text{prune}}(H_{1:t}) = H_{1:k}, \tag{7}$$

where $k < t$ is the divergence index.

4.2.3 Category 3: Attention Management

Cognitive Boosting. Even with complete information, models may face a reasoning gap in translating context into action. $\mathcal{O}_{\text{boost}}$ injects a Chain-of-Thought or sub-goal definition at the end of the context. This acts as a system hint, explicitly bridging the implicit logic required for the next step:

$$\begin{aligned} \mathcal{O}_{\text{boost}}(H) &= H \oplus z_{\text{thought}}, \\ z_{\text{thought}} &\sim P_{\text{CoT}}(\cdot|H). \end{aligned} \tag{8}$$

Key Anchoring. Addressing the Lost-in-the-Middle phenomenon, $\mathcal{O}_{\text{anchor}}$ exploits the Recency Bias of LLMs. It identifies global constraints or critical instructions c_{key} that may have been diluted by subsequent turns and copies them to the Active Attention Zone:

$$\mathcal{O}_{\text{anchor}}(H) = H \oplus \text{“[REMINDER]: ”} \oplus c_{key}. \tag{9}$$

4.2.4 The Identity Operator

Finally, we define $\mathcal{O}_{\text{NONE}}(H) = H$, which is selected when the Router determines that the current context requires no intervention.

4.3 Teacher-Guided Self-Evolving Training Paradigm

Training the model to effectively balance operator selection and context rewriting is non-trivial. To address this, we propose **Teacher-Guided Self-Evolving (TGSE)** Training Paradigm, a progressive training framework that transitions the model from supervised imitation to autonomous self-evolution. The training pipeline consists of three distinct phases:

Phase I: Supervised Initialization. To mitigate the instability of random exploration in the early stages, we cold-start both the Router policy π_θ and the Refactorer ϕ_ω with a small seed set $\mathcal{D}_{\text{seed}}$. Specifically, we employ a strong teacher model to generate high-quality supervision for routing decisions and corresponding refactored contexts $\hat{H}_t$. We then perform supervised fine-tuning on these teacher-labeled instances, yielding an initial router that can reliably identify when refactoring is needed and an initial refactorer that can execute basic context edits with high fidelity.

Phase II: Teacher-Guided Trajectory Rollout. We bootstrap high-quality corrective supervision with a teacher-in-the-loop rollout procedure. Given the current history H_t, the student Router π_θ first samples an intervention decision. To avoid propagating errors from an immature Refactorer, we then delegate the execution of the selected operator to a strong teacher model, producing a higher-fidelity refactored context $\hat{H}_{\text{teacher}}$. We finally perform hindsight verification by running the base solver on both H_t and $\hat{H}_{\text{teacher}}$ and measuring the resulting task outcome. We keep only the cases where teacher refactoring yields a clear improvement and add them as positive training instances for subsequent self-evolution.

Phase III: Autonomous Evolution. As the local Refactoring module ϕ_ω matures, we progressively decouple the system from the teacher. The framework enters a Bootstrapping mode, where the system samples and verifies trajectories using locally generated contexts $\hat{H}_{local}$. This facilitates the internalization of refactoring capabilities and enables closed-loop iteration.

Dynamic Data Synthesis Strategy. To ensure a balanced learning objective, the training data for self-evolution is dynamically composed of three distinct categories:

Corrective Instances. We form corrective pairs where the model fails under the raw history but succeeds after refactoring:

$$(H_t, y_{\text{fail}}) \rightarrow (\hat{H}_t, y_{\text{success}}). \tag{10}$$

These samples provide the most informative supervision for both modules: they encourage the Router to trigger refactoring when the current context exhibits drift or accumulated noise, and train the Refactorer to remove misleading or stale information that causes failure. Accordingly, we assign them a higher loss weight to emphasize failure-to-success transitions.

Compressive Instances. We create compressive pairs that preserve task success while reducing context length:

$$\begin{aligned}(H_t, y_{\text{success}}) \rightarrow (\hat{H}_t, y_{\text{success}}),\\ \text{s.t. } |\hat{H}_t| \ll |H_t|.\end{aligned} \tag{11}$$

These instances teach the model to retain only the information necessary for correct reasoning, improving efficiency via higher information density without degrading accuracy.

Regularization Instances. For contexts that are already clean and unambiguous, we include non-intervention examples:

$$(H_t, y_{\text{success}}) \rightarrow (\text{Action: None}). \tag{12}$$

They explicitly discourage unnecessary edits, preventing an over-refactoring tendency and stabilizing performance on easy or low-noise cases.

Algorithm 1 TGSE Training

Input: Seed set $\mathcal{D}_{\text{seed}}$, operators $\mathcal{O} \cup \{\text{NONE}\}$, feedback $R(\cdot)$
Phase I (Cold Start): Use a teacher to generate supervision on $\mathcal{D}_{\text{seed}}$; SFT Router π_θ and Refactorer ϕ_ω.
Phase II+ (Self-Evolution):
for each iteration **do**
  sample a task segment and obtain H_t
  sample $o_t \sim \pi_\theta(\cdot \mid H_t)$
  obtain $\hat{H} \leftarrow \text{TEACHER}(H_t, o_t)$ with prob. p_{teacher}, else $\hat{H} \leftarrow \phi_\omega(H_t, o_t)$
  compute $R(H_t)$ and $R(\hat{H})$
  if $R(\hat{H}) \geq R(H_t) + \delta$ **then**
    push into $\mathcal{B}_{\text{corr}}$
  else if $R(\hat{H}) \approx R(H_t)$ **and** $|\hat{H}| \ll |H_t|$ **then**
    push into $\mathcal{B}_{\text{comp}}$
  else
    push into $\mathcal{B}_{\text{reg}}$ (supervise NONE)
  end if
  sample minibatch from pools by fixed ratios and update π_θ, ϕ_ω
  anneal $p_{\text{teacher}} \downarrow$
end for=0

5 Experiments

In this section, we first introduce our experimental setup, and then report the main results, ablation studies, and efficiency analysis to comprehensively validate the effectiveness of the proposed ACR framework. More details of the experiments can be seen in Appendix A.

5.1 Experimental Setups

Datasets. We evaluate our framework on *multi-turn QA* benchmarks that stress long-range dependency tracking and evidence aggregation. Following prior agent-style QA settings, we build the training set by merging **Natural Questions (NQ)** (single-hop) and **HotpotQA** (multi-hop) (Jin et al., 2025; Zheng et al., 2025), and assess generalization on a suite of **seven** QA datasets: single-hop QA (**NQ** (Kwiatkowski et al., 2019), **TriviaQA** (Joshi et al., 2017), and **PopQA** (Mallen et al., 2023)) and multi-hop QA (**HotpotQA** (Yang et al., 2018), **2Wiki** (Ho et al., 2020), **MusiQue** (Trivedi et al., 2022), and **Bamboogle** (Press et al., 2023)). See Appendix A.1 for the full list and dataset statistics. We use **Exact Match (EM)** as the metric.

Baselines. We compare our method against representative baselines across several paradigms, including **prompting** (DIRECT INFERENCE, COT (Wei et al., 2022), IRCOT (Trivedi et al., 2023)), **SFT** (Chung et al., 2024), **retrieval-augmented QA** (DRAGIN (Su et al., 2024), DIOR (Guo et al., 2025b), SEAKR (Yao et al., 2025)), **context compression** (RECOMP (Xu et al., 2024)), **external memory** (HIPPORAG (Jimenez Gutierrez et al., 2024), ITER-RETGEN (Shao et al., 2023)), and **RL-style search training** (R1-INSTRUCTION (Guo et al., 2025a), SEARCH-R1 (Jin et al., 2025), STEPSEARCH (Zheng et al., 2025)); full baseline details are deferred to Appendix A.2.

Train Details. To ensure a controlled compari-

Table 1: Main results (EM, %) on seven QA benchmarks (single-hop and multi-hop). We compare ACR with baselines under the same retriever and backbone. † denotes in-domain datasets and ⋆ denotes out-of-domain datasets.

Type	Method	Single-Hop QA			Multi-Hop QA			
		NQ†	TriviaQA⋆	PopQA⋆	HotpotQA†	2Wiki⋆	MuSiQue⋆	Bamboogle⋆
Prompt	Direct inference	13.40	40.80	14.00	18.30	25.00	3.10	12.00
	CoT	4.80	18.50	5.40	9.20	11.10	2.20	23.20
	IRCoT	22.40	47.80	30.10	13.30	14.90	7.20	22.40
SFT	SFT	31.80	35.40	12.10	21.70	25.90	6.60	11.20
RAG	DRAGIN	23.20	42.00	–	23.20	22.00	–	–
	DioR	26.20	52.30	–	27.40	26.60	–	–
	SEAKR	25.60	54.40	–	27.90	30.20	–	–
Compression	RECOMP (Flan-UL2-20B)	36.60	58.99	–	30.40	–	–	–
Mem.	HippoRAG (GPT-3.5)	–	–	–	**45.70**	**47.70**	21.90	–
	ITER-RETGEN	–	–	–	45.20	35.50	**25.90**	**40.00**
RL Training	R1-Instruction	21.00	44.90	17.10	20.80	27.50	6.00	19.20
	Search-R1	**39.30**	**61.00**	**39.70**	37.00	40.10	14.60	36.80
	StepSearch	–	–	–	38.60	36.60	22.60	**40.00**
SFT	**ACR (Ours)**	36.41	56.86	36.04	35.10	34.32	16.67	36.36

son, we adopt the same retriever setting as Search-R1 and use E5 as the retriever. We use **Qwen-2.5-7B-Instruct** as the downstream reasoner, GPT-5.2 as teacher model and instantiate both the **Router** and **Refactorer** on the same backbone for parameter sharing and fair capacity matching. We train the Router and Refactorer with parameter-efficient adapters under the proposed TGSE training paradigm, and implement all iterative updates using **LLaMA-Factory**. Hyperparameter settings are reported in Appendix A.3.

5.2 Experimental Results

In this section, we report our main experimental results to validate the effectiveness of ACR across diverse datasets. We further study the contribution of each component and analyze the efficiency of our approach. Additional experimental results and extended analyses are provided in the Appendix A.

5.2.1 Overall Experiments

Table 1 reports EM results of ACR on seven QA benchmarks. We compare ACR with prompting (Direct/CoT/IRCoT), SFT, RAG variants, compression/external-memory methods, and RL-style search training. The results support four observations. **(1) Consistent gains over conventional paradigms.** ACR yields stable improvements over prompting and vanilla SFT on all datasets. In particular, it improves SFT from 31.80→36.41 on NQ, 35.40→56.86 on TriviaQA, 12.10→36.04 on PopQA, and 21.70→35.10 on HotpotQA, suggesting that need-driven history refactoring is broadly effective for both single-hop and multi-hop QA. **(2) Interpreting compression/external-memory baselines.** Several compression/memory baselines report strong results on multi-hop settings, but their advantage is partly driven by stronger backbones in their default configurations (e.g., RECOMP with Flan-UL2-20B; HippoRAG with GPT-3.5). **(3) Low-cost competitiveness vs. RL-style training.** RL-based search training remains highly competitive. Nevertheless, ACR narrows the gap with substantially lower training overhead: it trains only an external routing/refactoring controller using 3.8K supervised instances, compared with 170K for Search-R1 and 19K for StepSearch, while avoiding online rollouts, reward engineering, and policy optimization. Despite this lightweight setup, ACR stays close to Search-R1 and surpasses it on MuSiQue (16.67 vs. 14.60), demonstrating favorable cost–performance trade-offs. **(4) Modularity and generality.** Notably, ACR updates only the external controller while keeping the underlying reasoner fixed, yet consistently improves performance across diverse QA benchmarks, highlighting the generality of plug-in context refactoring.

5.2.2 Efficiency Analysis

We evaluate the efficiency of our approach by analyzing the average number of generated tokens per turn, as shown in Figure 3. Compared to Reinforce-

Table 2: Ablation results of ACR (EM, %). We disable the Router or Refactorer to assess their contributions across QA benchmarks. Where "Base" refers to using Qwen2.5-7B-instruct as the router and refactorer.

Variant	Single-Hop QA			Multi-Hop QA			
	NQ†	TriviaQA⋆	PopQA⋆	HotpotQA†	2Wiki⋆	MuSiQue⋆	Bamboogle⋆
Base	29.56	47.49	22.16	18.43	18.77	8.48	15.74
w/o Router	31.03	48.34	25.32	21.28	20.78	9.78	23.45
w/o Refactorer	34.38	47.62	30.43	24.53	23.97	10.65	27.56
Ours	**36.41**	**56.86**	**36.04**	**35.10**	**34.32**	**16.67**	**36.36**

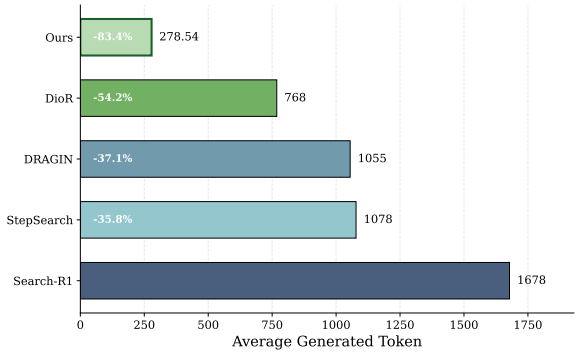

Figure 3: Comparison of average generated tokens across different methods.

ment Learning (RL) based methods such as **Search-R1**, our approach demonstrates a significant advantage in computational efficiency. While Search-R1 incurs a high cost averaging **1678** tokens, our method reduces this consumption to **278.54** tokens (**-83.4%**). Although our method may slightly trail RL approaches in raw performance metrics, the marginal difference is outweighed by this massive gain in efficiency, making it a more practical solution for resource-constrained environments.

Furthermore, when compared to traditional methods like **DioR** and **DRAGIN**, our approach proves to be both **stronger and more efficient**. We not only achieve lower token consumption but also deliver superior reasoning capabilities, demonstrating that our method effectively eliminates redundant steps without compromising solution quality.

5.2.3 Ablation Study

Table 2 reports ablations of ACR. We additionally include a **Base** controller baseline, where both the Router and Refactorer are instantiated by the **Qwen2.5-7B-Instruct** (prompt-only, *without* training). Disabling either module yields clear degradations across both single-hop and multi-hop benchmarks. The largest drops come from removing the **Router**: performance decreases by **13.82** EM on HotpotQA (35.10→21.28), **13.54** EM on 2Wiki (34.32→20.78), and **12.91** EM on Bamboogle (36.36→23.45). This indicates that *when-to-intervene* (online drift detection and intervention triggering) is crucial, especially for multi-hop reasoning where early deviations can propagate non-locally.

The **Refactorer** is also indispensable: removing it still causes substantial losses (e.g., **10.57** EM on HotpotQA and **9.24** EM on TriviaQA), suggesting that accurate detection alone is insufficient without effective *how-to-fix* execution that denoises and restructures the history. Overall, ACR reaches its best performance only when the Router and Refactorer operate in a closed loop, coupling diagnosis with corrective refactoring.

6 Conclusion

In this paper, we first investigate the limitations of large language models in multi-turn dialogue as well as prior work that has attempted to address these limitations. However, current methods continue to face significant challenges: **(1)Contextual Inertia** and **(2)State Drift**. To overcome these problems, we propose an innovative framework **A**daptive **C**ontext **R**efactoring **(ACR)**, that actively manages the evolving history instead of passively concatenating it. ACR uses an external controller to monitor the interaction history, select a refactoring operator, and rewrite the context into a cleaner, task-relevant form, thereby decoupling context management from task reasoning. We further introduce a teacher-guided self-evolving training scheme to iteratively improve the router and refactorer. Experiments on single-hop and multi-hop QA benchmarks show consistent gains over strong baselines. In future work, we will internalize refactoring skills into a single unified model.

Limitations

Our framework improves multi-turn reasoning by introducing an external controller to monitor and refactor the dialogue history when needed. This design, however, comes with additional deployment overhead. In practice, ACR requires extra modules and thus increases system complexity, memory footprint, and inference latency, even though it can reduce the token budget of the downstream reasoner. As a result, the overall efficiency trade-off may vary across hardware settings and latency constraints. A promising direction is to further internalize refactoring capability into a single model, e.g., by distillation or unified training, so that context refactoring and task reasoning can be performed within one model without relying on an external controller.

In addition, our current evaluation is mainly conducted on QA-style multi-turn reasoning. While these benchmarks capture long-range dependency tracking and factual consistency, they do not fully represent agentic settings that require long-horizon planning, tool use, and interaction with dynamic environments. Therefore, it remains unclear how ACR will behave under more complex agentic workloads, where errors can compound through actions and observations. In future work, we plan to conduct a more comprehensive study on multi-turn agent tasks and interactive environments, and refine the framework to better accommodate task-specific constraints and feedback signals.

Ethics Statement

This work utilizes publicly available standard benchmark datasets for evaluation and training, including Natural Questions, HotpotQA, TriviaQA, PopQA, 2WikiMultiHopQA, MuSiQue, and Bamboogle. These datasets are primarily derived from public knowledge sources (e.g., Wikipedia) and do not contain personally identifiable information (PII) or sensitive personal data.

The training process employs a Teacher-Guided Self-Evolving paradigm that relies on synthetic supervision generated by LLMs, without involving new human subject experiments or crowdsourced annotation. Furthermore, the proposed Adaptive Context Refactoring framework aims to enhance the reliability and factual consistency of multi-turn dialogue systems by actively mitigating hallucinations, thereby contributing to the development of safer and more robust AI systems.

Acknowledgment

We acknowledge the support of the "Pioneer" and "Leading Goose" R&D Program of Zhejiang (No. 2026C02A1236) and the National Natural Science Foundation of China under Grant (No. 62577050).

References

Sanghwan Bae, Donghyun Kwak, Soyoung Kang, Min Young Lee, Sungdong Kim, Yuin Jeong, Hyeri Kim, Sang-Woo Lee, Woomyoung Park, and Nako Sung. 2022. Keep me updated! memory management in long-term conversations. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3769–3787, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024. Longlora: Efficient fine-tuning of long-context large language models. In *The International Conference on Learning Representations (ICLR)*.

Yu-Neng Chuang, Tianwei Xing, Chia-Yuan Chang, Zirui Liu, Xun Chen, and Xia Hu. 2024. Learning to compress prompt in natural language formats. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7756–7767, Mexico City, Mexico. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR 2024. ACM.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: fast and memory-efficient exact attention with io-awareness. In *Proceedings of the 36th International Conference*

on Neural Information Processing Systems, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Nouha Dziri, Sivan Milton, Mo Yu, Osmar Zaiane, and Siva Reddy. 2022. On the origin of hallucinations in conversational models: Is it the datasets or the models? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5271–5285, Seattle, United States. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hanghui Guo, Jia Zhu, Shimin Di, Weijie Shi, Zhangze Chen, and Jiajie Xu. 2025b. DioR: Adaptive cognitive detection and contextual retrieval optimization for dynamic retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2953–2975, Vienna, Austria. Association for Computational Linguistics.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Zhengbao Jiang, Luyu Gao, Zhiruo Wang, Jun Araki, Haibo Ding, Jamie Callan, and Graham Neubig. 2022. Retrieval as attention: End-to-end learning of retrieval and reading within a single transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2336–2349.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

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 *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711.

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.

Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. 2024. DRAGIN: Dynamic retrieval augmented generation based on the real-time information needs of large language models. In *Proceedings*

of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12991–13013, Bangkok, Thailand. Association for Computational Linguistics.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 10014–10037.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. Recomp: Improving retrieval-augmented lms with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Zijun Yao, Weijian Qi, Liangming Pan, Shulin Cao, Linmei Hu, Liu Weichuan, Lei Hou, and Juanzi Li. 2025. SeaKR: Self-aware knowledge retrieval for adaptive retrieval augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27022–27043, Vienna, Austria. Association for Computational Linguistics.

Biao Zhang, Yunwei Chen, and Daobin Gao. 2025. Development characteristics and topic analysis of international research on chatgpt. *DATA INTELLIGENCE*, 7(4):1192–1217.

Xuhui Zheng, Kang An, Ziliang Wang, Yuhang Wang, and Yichao Wu. 2025. StepSearch: Igniting LLMs search ability via step-wise proximal policy optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21805–21830, Suzhou, China. Association for Computational Linguistics.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731.

Jia Zhu, Hanghui Guo, Weijie Shi, Zhangze Chen, and Pasquale De Meo. 2025a. Radio: Real-time hallucination detection with contextual index optimized query formulation for dynamic retrieval augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 26129–26137.

Nana Zhu, Caihai Zhu, Qingfu Zhu, and Yuanxing Liu. 2025b. Style rewriter: A three-stage approach for stylistic generation of open-domain conversational responses. *DATA INTELLIGENCE*, 7(2):397–415.

This appendix provides supplementary material to ensure the clarity, depth, and reproducibility of our work. It is structured as follows:

A Experiments Details

A.1 Data Statistics

We construct a training pool by merging two QA sources: Natural Questions (NQ) and HotpotQA, resulting in **169,615** training instances in total, with **79,168** (46.7%) from NQ and **90,447** (53.3%) from HotpotQA. For evaluation, we curate a seven-dataset benchmark suite with **51,713** examples, covering both single-hop and multi-hop QA: PopQA **14,267** (27.6%), 2WikiMultiHopQA **12,576** (24.3%), TriviaQA **11,313** (21.9%), HotpotQA **7,405** (14.3%), NQ **3,610** (7.0%), MuSiQue **2,417** (4.7%), and Bamboogle **125** (0.2%). Notably, although the training pool is large, our self-evolving training uses only a small subset: we start with a cold-start set of **400** examples, and then sample **200** examples per iteration for **17** subsequent iterations (18 rounds in total), yielding **400 + 17×200 = 3,800** training examples overall (about **2.24%** of the full training pool). This design allows us to study the sample efficiency of ACR under a strictly limited supervision budget.

Split	Dataset / Source	#Examples	Share (%)
Training	NQ	79,168	46.7
Training	HotpotQA	90,447	53.3
Eval	PopQA	14,267	27.6
Eval	2WikiMultiHopQA	12,576	24.3
Eval	TriviaQA	11,313	21.9
Eval	HotpotQA	7,405	14.3
Eval	NQ	3,610	7.0
Eval	MuSiQue	2,417	4.7
Eval	Bamboogle	125	0.2
Training budget used in TGSE		3,800	2.24

Table 3: Dataset statistics. The training pool is formed by merging NQ and HotpotQA, while evaluation spans seven QA benchmarks.

A.2 Baselines

For single-hop and multi-hop QA, we compare our approach with a diverse set of competitive baselines:

(1) **Prompting baselines:** PROMPT, COT, and IRCOT, which rely purely on in-context prompting (with IRCoT further incorporating iterative retrieval into the reasoning trace) without updating model parameters.

(2) **Supervised fine-tuning:** SFT, a task-adapted baseline trained with standard supervised learning.

(3) **Retrieval-augmented QA:** DRAGIN, DIOR, and SEAKR, which dynamically trigger retrieval and augment generation with external evidence.

(4) **Context compression:** RECOMP, which improves long-context efficiency by selectively compressing and augmenting contexts.

(5) **External memory methods:** HIPPORAG and ITER-RETGEN, which leverage long-term memory structures / iterative retrieval-generation to better support multi-hop evidence aggregation.

(6) **RL-style search training:** R1-INSTRUCT, SEARCH-R1, and STEPSEARCH, which optimize search-augmented reasoning policies via reinforcement learning-style training signals.

A.3 Training Details

Backbone models. The Router and Refactoring modules are implemented as lightweight LoRA adapters attached to an external controller model. For the downstream solver (reasoner), we use QWEN-2.5-7B-INSTRUCT as a frozen backbone. We do not update the solver parameters during training. Unless otherwise specified, we decode with temperature 0.7 and set the maximum generation length to 8192 tokens.

Training hyperparameters. We employ the AdamW optimizer with a learning rate of 2×10^{-4} and a per-device batch size of 4. For parameter-efficient fine-tuning, we utilize LoRA with rank $r = 16$ and $\alpha = 32$. The training process is set to run for 3 epochs per iteration. To prevent overfitting, we apply an early-stopping criterion on the validation set with a patience of 3 iterations and a minimum improvement threshold (min_delta) of 0.001. Regarding the TGSE knobs: we set the early stopping delta $\delta = 0.001$. Note that the pool ratios and p_{teacher} annealing schedule are not explicitly defined in the provided configuration and may require manual specification based on the implementation details.

Hardware. All experiments are conducted on 2× NVIDIA A100 GPUs (80GB each). We use LoRA training to reduce the memory footprint and accelerate training.

A.4 Training Loss

Training dynamics. Figure 4 and Figure 5 illustrate the training loss trajectories of the Router

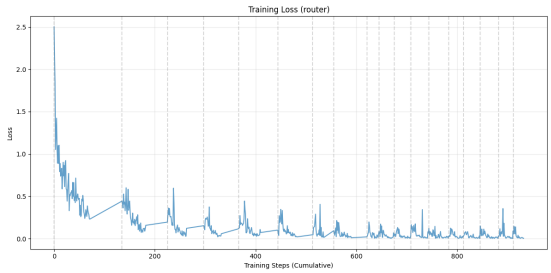

Figure 4: Training loss of the Router (LoRA) across TGSE rounds.

Figure 5: Training loss of the Refactorer (LoRA) across TGSE rounds.

and Refactorer (LoRA) under TGSE. Both modules exhibit a clear convergence pattern: the loss drops sharply during the cold-start stage, indicating that a small amount of supervision is sufficient to learn effective routing decisions and basic refactoring behaviors. Across subsequent self-evolving rounds, the curves show short-lived, iteration-aligned spikes (marked by dashed vertical lines), which we attribute to mild distribution shifts when newly sampled data are introduced. Importantly, these perturbations are quickly absorbed within a few update steps, and the loss returns to a low regime. Compared to the Router, the Refactorer shows a larger variance in early rounds, consistent with the higher difficulty and heterogeneity of generation-style rewriting objectives; however, its fluctuations diminish over time and stabilize near zero, with only sparse spikes on harder conflict-heavy cases. Overall, the loss trends suggest stable optimization and fast adaptation to round-wise data refreshes.

B Implementation Details

B.1 System Overview

Our framework realizes a *self-evolving* training loop that repeatedly alternates between **trajectory generation** and **incremental fine-tuning**. At inference time, we augment a standard agent with an explicit **controller** consisting of a **Router** (drift diagnosis) and a **refactorer** (context refactoring). Given the evolving interaction history H_t, the Router decides whether refactoring is needed and selects an operator $o_t \in \mathcal{O} \cup \{\text{NONE}\}$. If $o_t \neq \text{NONE}$, the refactorer produces a refactored context $\tilde{H}_t$, which is then used to condition the Actor for the next action. This design separates *context management* from *task solving*, enabling modular training and controlled interventions.

B.2 Controller I/O Protocol

History-only diagnosis. To avoid trivially conditioning on the current query and to encourage robust drift detection, the Router receives only the accumulated history H_t (and the task description) rather than the current step query. It outputs a compact decision in a structured form: a binary drift flag and an operator choice. This protocol makes the Router a *context monitor* rather than a solver.

Refactoring as replaceable state. The refactorer outputs a single refactored context block $\tilde{H}_t$ that is directly replaceable as the "previous context" buffer. We enforce a strict output contract (no extra commentary) to prevent hidden leakage into the Actor prompt and to ensure that refactoring is auditable.

B.3 Self-Evolving Data Generation

Cold-start supervision. We initialize learning from a small seed set produced by a strong teacher model. The teacher provides (i) operator decisions for the Router and (ii) refactored contexts for the refactorer. This stage stabilizes early training by preventing degenerate policies (e.g., always selecting a single operator).

Evolution via staged locality. After cold start, we iteratively improve the controller by generating new trajectories with increasingly local components: in the first evolution iteration, we run a *hybrid* setting where the Router is local (student) while the refactorer remains teacher-guided. From the second iteration onward, both Router and refactorer operate locally. This staged schedule mitigates compounding errors: early on, we distill high-quality refactoring behaviors before relying fully on local refactoring.

Branch forking for supervision diversity. When the Router detects drift, we optionally fork an environment state at the refactoring step and continue execution along two branches: a baseline branch (without refactoring) and a refactored

branch. Forking increases the diversity of corrective examples and allows the system to observe counterfactual outcomes under different context states. To control computational growth, we bound the maximum refactoring depth per episode and cap the refactoring budget of each trajectory.

B.4 Training Objective and Optimization

We train Router and refactorer as **separate** parameter-efficient adapters over a shared base model. Both modules are optimized with supervised fine-tuning using the collected JSONL pairs. During evolution, we perform *incremental* adapter updates by initializing from the previous iteration's adapter, which enables continual improvement without retraining from scratch.

Early stopping per module. Since operator selection and context rewriting may converge at different speeds, we track their validation losses independently. We stop updating a module if its loss fails to improve beyond a minimum margin for several iterations, while allowing the other module to continue evolving. This prevents overfitting and unnecessary computation once one component saturates.

B.5 Engineering Choices for Stability and Throughput

Efficient local inference with dynamic adapters. For fully local evolution, we use an offline inference backend that supports dynamic LoRA switching. This enables updating Router/refactorer adapters between iterations without reloading the base model, substantially reducing iteration overhead.

B.6 Reproducibility Details

Configuration. All hyperparameters and system choices are specified in a single YAML configuration. We export the resolved configuration alongside training/validation logs for each run.

Data and checkpointing. For each phase/iteration, we save the Router and refactorer datasets separately. We also persist a lightweight checkpoint containing: current iteration, adapter paths, dataset paths, sample counts, and a record of used tasks for de-duplication. This allows resuming evolution without repeating data collection and helps prevent accidental train-eval leakage.

C Prompts

We list the prompt templates used by the search agent, router, and refactorer modules. Please refer to our code base for more details.

Placeholders. We use the following placeholders in our prompt templates: `{task_description}` (the task goal or user query), `{history}` (the raw interaction history between the agent and the environment/user), `{previous_context}` (the previously refactored context; set to `None` if empty), `{step_count}` (the number of steps taken so far in the search loop), and `{memory_context}` (a structured search memory that concatenates prior `<search>...</search>` queries and the corresponding `<information>...</information>` results). All outputs follow strict structural contracts to enable deterministic parsing.

C.1 Search Prompts

We standardize the search-augmented reasoning procedure with a unified SEARCH-AGENT prompt. At each step, the model must first write its internal reasoning enclosed in `<think>...</think>`, and then emit exactly one action: either (i) issue a web query using `<search>...</search>` when external evidence is needed, or (ii) return the final answer using `<answer>...</answer>` without additional explanations. For multi-step interactions, we additionally provide the accumulated step count and a structured memory context, where previous queries and retrieved evidence are explicitly tagged with `<search>` and `<information>` to support traceable evidence aggregation and prevent mixing retrieval with answering in the same step.

Search Prompt Template

You are an expert agent tasked with answering the given question step-by-step.
Your question: {task_description}
(Optional) Prior to this step, you have already taken {step_count} step(s). Below is the interaction history, where `<search>...</search>` wraps your past search queries and `<information>...</information>` wraps the corresponding results returned by the external search engine: {memory_context}
Now it's your turn to respond for the current step.
You should first conduct a reasoning process. This process MUST be enclosed within `<think>` `</think>` tags. After completing your reasoning, choose **only one** of the following actions (do not perform both): (1) If you lack external knowledge, call a search engine using: `<search> your query </search>`. (2) If you have enough knowledge to answer confidently, provide the final answer using: `<answer> ... </answer>` (no additional explanation).

C.2 Router Prompt

C.3 Refactorer Prompts

Operator templates. Given a Router-selected operator o, we instantiate an operator-specific user prompt by filling the placeholders `{task_description}`, `{history}`, and `{previous_context}`.

We enforce strict output contracts for reliable deployment. For the Router, we parse the output as JSON and validate all required fields, in particular `drift_detected` and `selected_operator` against the predefined operator set. If parsing or validation fails, we apply a conservative fallback by setting `drift_detected=false` and `selected_operator="none"`, and pass the raw history forward. For the Refactorer, we extract the content within `<summary>...</summary>` tags; if tags are missing or the extracted summary is empty, we fall back to using the raw model output as the refactored context. These safeguards prevent error propagation and ensure the pipeline never silently proceeds with an ill-formed refactoring.

Router System Prompt

You are a Context Monitor for multi-turn dialogue LLMs.
Your job is to analyze the accumulated history and detect context drift, i.e., cases where the history becomes noisy, misleading, inconsistent, or excessively long and may harm the LLM's next decision.
Important constraint: You will receive only the history context, not the current query. Decide whether refactoring would help with the next action.
Available operators: 1) state_abstract: compress verbose history into a high-level state snapshot. Use when the key state is buried in details. 2) noise_filter: remove irrelevant or redundant content. Use when the history contains off-topic or repeated text. 3) fact_rectify: correct contradictions or factual errors. Use when the history conflicts with the current state. 4) path_prune: remove repeated failures or loops. Use when the history shows circular attempts. 5) cognitive_boosting: inject a short guiding thought to refocus. Use when the LLM is confused about the goal or next step. 6) attention_anchor: move or copy critical constraints to the end for recall. Use when important requirements are being overlooked. 7) none: no refactoring needed. Use when the history is clean and focused.
Output format (strict): Return only a valid JSON object with the following fields: {"analysis": "<brief 1-2 sentence explanation>", "drift_detected": <true or false>, "selected_operator": "<operator_name>"}
Rules: - If drift_detected is false, selected_operator must be "none". - If drift_detected is true, selected_operator must be one of the six active operators. - Be conservative: only flag drift when it is likely to impact reasoning. - Output only the JSON object, nothing else.

Figure 6: Router system prompt. Markdown markers are removed; the output contract is strictly JSON.

Router User Message Template

Task description: {task_description}
History context (analyze for drift): {history}
Instruction: Analyze the history and output your assessment as a JSON object.

Figure 7: Router input template.

Refactorer Shared System Prompt

You are a Context Refactoring Engine for multi-turn dialogue LLMs.
Your job is to transform the provided history to improve the LLM's next decision by applying a specified transformation operator.
Core principles: 1) Preserve critical information needed for task completion. 2) Maintain coherence: the refactored context must be logically consistent and self-contained. 3) Enable progress: the result should support better decisions going forward. 4) Be conservative: when uncertain, preserve rather than remove.
Output rules: 1) Output only the refactored context within <summary> </summary> tags. 2) The refactored context must be directly usable as the new history. 3) Do not include any explanation or meta-commentary. 4) Do not add information that was not present in the original context. 5) Aim for meaningful compression while keeping critical information.

Figure 8: Refactorer shared header used across all operators.

Operator Prompt: state_abstract

Role: You are a precise text processing engine specialized in state abstraction.
Operator name: State Abstraction
Objective: Compress the interaction history into a concise state snapshot.
Transformation logic: 1) Identify the net results of actions in history. 2) Remove intermediate steps that no longer matter. 3) Replace detailed sequences with a compact description of the current physical/logical state. 4) Keep the most recent observation and any critical discoveries.
Key principles: - Focus on what has been achieved, not how it was achieved. - Track inventory changes. - Track environment state changes. - Preserve discovered constraints and rules.
Current task: {task_description}
Previous refactored context: {previous_context}
Input context (raw interaction history): {history}
Output format: Return only: <summary> ... state snapshot ... </summary>

Figure 9: Operator prompt for STATE_ABSTRACT.

Operator Prompt: noise_filter

Role: You are a precise text processing engine specialized in noise filtration.
Operator name: Noise Filtration
Objective: Remove irrelevant noise while preserving useful information.
Transformation logic: 1) Identify segments orthogonal to the task. 2) Delete such segments entirely. 3) Ensure the remaining text is coherent and temporally consistent.
Remove: - Repeated identical observations. - Verbose descriptions that add no information. - Failed actions that provide no new constraints. - Navigation steps that do not change state.
Keep: - State-changing actions and outcomes. - New discoveries. - Constraint-bearing error messages. - The most recent observation.
Current task: {task_description}
Previous refactored context: {previous_context}
Input context (noisy history): {history}
Output format: Return only: <summary> ... filtered context ... </summary>

Figure 10: Operator prompt for NOISE_FILTER.

Operator Prompt: fact_rectify

Role: You are a precise text processing engine specialized in fact rectification.
Operator name: Fact Rectification
Objective: Identify and correct inconsistencies or contradictions in the interaction history while preserving correct content.
Trusted signals: - The current task description defines the goal. - The most recent observation reflects the true current state. - Successful actions are factual evidence; failed actions reveal constraints.
Transformation logic: 1) Locate statements in the history that conflict with the current observed state. 2) Detect logical inconsistencies. 3) Edit only the minimal conflicting spans to match trusted signals. 4) Preserve all correct parts of the history unchanged. 5) Do not invent new facts that are not supported by the task or observations.
Common issues to fix: - Incorrect inventory tracking. - Wrong location assumptions. - Misremembered action outcomes. - Contradictory state descriptions.
Current task: {task_description}
Previous refactored context: {previous_context}
Input context: {history}
Output format: Return only: <summary> ... rectified context ... </summary>

Figure 11: Operator prompt for FACT_RECTIFY.

Operator Prompt: path_prune

Role: You are a precise text processing engine specialized in path pruning.
Operator name: Path Pruning
Objective: Truncate the history to remove failed branches and repetitive loops, while preserving any useful discoveries.
Transformation logic: 1) Identify where the interaction begins to loop, stall, or repeatedly fail. 2) Recognize loop patterns such as: - trying the same action multiple times with the same failure, - repeatedly searching the same locations without new findings, - back-and-forth navigation returning to an unchanged state. 3) Delete the looping or dead-end portion. 4) Preserve any new information discovered, even within the failed branch. 5) End the context at a clean decision point so the LLM can attempt a new plan.
Pruning criteria: - Remove sequences of three or more similar failed actions. - Remove back-and-forth navigation that returns to the same state. - Remove repeated searches of empty containers or rooms. - Keep: discoveries, constraints, and the latest valid state summary.
Current task: {task_description}
Previous refactored context: {previous_context}
Input context (history with potential loops/failures): {history}
Output format: Return only: <summary> ... pruned context ending at a clean decision point ... </summary>

Figure 12: Operator prompt for PATH_PRUNE.

Operator Prompt: cognitive_boosting

Role: You are a precise text processing engine specialized in cognitive reinforcement.
Operator name: Cognitive Reinforcement
Objective: Insert a short guiding directive to refocus the LLM and improve the next decision, without changing factual content.
Transformation logic: 1) Identify where the LLM becomes confused, inefficient, or stuck in the history. 2) Determine an actionable next sub-goal based on the task and the current state. 3) Insert a directive formatted exactly as: [Thought]: ... 4) The directive must be specific and actionable (what to do next), but must not add unsupported facts.
Reinforcement strategies: - If stuck: recommend unexplored locations or a different interaction strategy. - If confused: restate the immediate sub-goal clearly. - If inefficient: suggest a more direct plan. - If close to completion: highlight the remaining required steps.
Placement rule: Insert the [Thought] directive near the end of the refactored context, right before the most recent situation description, so it is salient for the next action.
Current task: {task_description}
Previous refactored context: {previous_context}
Input context: {history}
Output format: Return only: <summary> ... refactored context ...
[Thought]: ... (one or two sentences, actionable) </summary>

Figure 13: Operator prompt for COGNITIVE_BOOSTING.

Operator Prompt: attention_anchor

Role: You are a precise text processing engine specialized in attention anchoring.
Operator name: Attention Anchoring
Objective: Move or copy critical information to the end of the context so it remains in the active attention region for the next action.
Transformation logic: 1) Identify critical information mentioned earlier but essential for completing the task. 2) Typical critical information includes: task objective, constraints, inventory, discovered key locations/items, partial progress, and what has already been searched. 3) Append a final section named [KEY INFO] at the very end of the context. 4) Ensure the [KEY INFO] section is the last content before the model generates the next action. 5) Do not add new facts; only restate or re-organize information that exists in the input.
What to anchor: - The main task objective. - Current inventory. - Locations already searched. - Discovered constraints or rules. - Partial progress and remaining steps.
Current task: {task_description}
Previous refactored context: {previous_context}
Input context (history with potentially forgotten information): {history}
Output format: Return only: <summary> ... refactored context ...
[KEY INFO]: - Task: ... - Current inventory: ... - Searched locations: ... - Constraints: ... - Next logical step: ... </summary>

Figure 14: Operator prompt for ATTENTION_ANCHOR.