

CoDA: A Context-Decoupled Hierarchical Agent with Reinforcement Learning

Anonymous Author(s)

Abstract

Large Language Model (LLM) agents trained with reinforcement learning (RL) show great promise for solving complex, multi-step tasks. However, their performance is often crippled by "Context Explosion", where the accumulation of long text outputs overwhelms the model's context window and leads to reasoning failures. To address this, we introduce CoDA, a Context-Decoupled hierarchical Agent, a simple but effective reinforcement learning framework that decouples high-level planning from low-level execution. It employs a single, shared LLM backbone that learns to operate in two distinct, contextually isolated roles: a high-level Planner that decomposes tasks within a concise strategic context, and a low-level Executor that handles tool interactions in an ephemeral, isolated workspace. We train this unified agent end-to-end using PECO (Planner-Executor Co-Optimization), a reinforcement learning methodology that applies a trajectory-level reward to jointly optimize both roles, fostering seamless collaboration through context-dependent policy updates. Extensive experiments demonstrate that CoDA achieves significant performance improvements over state-of-the-art baselines on complex multi-hop question-answering benchmarks, and it exhibits strong robustness in long-context scenarios, maintaining stable performance while all other baselines suffer severe degradation, thus further validating the effectiveness of our hierarchical design in mitigating context overload. Our code is available at this anonymous repository.

Keywords

Large Language Model, Context Engineering, Reinforcement Learning, Hierarchical Agent

1 Introduction

LLM agents have emerged as a new frontier in autonomous problem-solving, demonstrating their capability to tackle complex, multi-step tasks by interacting with external tools. To unlock their full potential, recent studies have focused on training these agents via reinforcement learning (RL) to autonomously learn reasoning and tool-use strategies [9, 32]. This approach, where agents learn from outcomes in interactive environments, holds the promise of creating truly adaptive and intelligent systems.

Despite their promising results, we identify a core limitation that lies in the current retrieval-augmented reasoning paradigm,

which we term **Accumulated Context Explosion**. This refers to the phenomenon that *as tasks grow in complexity, requiring more tool interactions, LLMs need to invoke external tools multiple times to query answers, which results in lengthy tool retrieval outputs*. This places significant demands on the agent's context management, since previous approaches have primarily concatenated retrieval results directly into the context; however, the limited context window of current large models restricts their reasoning capabilities when dealing with lengthy inputs. For example, we have observed in our experiments that when each retrieval output reaches around 4000 tokens (see 4.4.1), the LLM model's performance deteriorates significantly. Specifically, LLMs fail to access relevant information in the middle of long contexts[17]. Even switching to models with larger context windows fails to alleviate this issue.

Our core insight is that overcoming this cognitive overload requires a framework that not only structurally separates high-level planning from low-level execution, but is also holistically learnable. Inspired by the classic "divide and conquer" principle, we propose **CoDA** (Context-Decoupled Hierarchical Agent with Reinforcement Learning), a novel framework that we illustrate in Figure 1, which operationalizes this concept. It employs a single, shared LLM that learns to operate in two distinct, contextually-isolated roles:

- (1) **Planner**: The Planner operates at a high level of abstraction, learning to decompose a complex request into a sequence of strategic, executable sub-tasks. It accepts the result from the Executor and plans the next step sequentially.
- (2) **Executor**: For each sub-task, the same underlying model acts as the Executor. It functions within an ephemeral and isolated context to perform detailed reasoning, tool calls, and information synthesis related to the assigned sub-task.

Critically, a **single, shared model** embodies both the Planner and Executor roles. This unified architecture enables **end-to-end training via reinforcement learning (RL)**, allowing for the coordinated optimization of both planning and execution and fostering seamless collaboration between the two roles.

We make the following contributions in this paper:

- We propose CoDA, a novel hierarchical agent framework that mitigates context overload by decoupling high-level strategic planning from low-level task execution. This is achieved by maintaining role-specific contexts for a Planner and an Executor, embodied by a single, shared LLM.
- We introduce a Planner-Executor Co-Optimization (PECO) training methodology, an end-to-end reinforcement learning approach that jointly optimizes both roles. By applying a unified trajectory-level reward to context-dependent policy updates, PECO fosters seamless collaboration and mutual adaptation between the Planner and Executor.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WSDM '26, Boise, Idaho, USA

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2026/02
<https://doi.org/XXXXXXX.XXXXXXX>

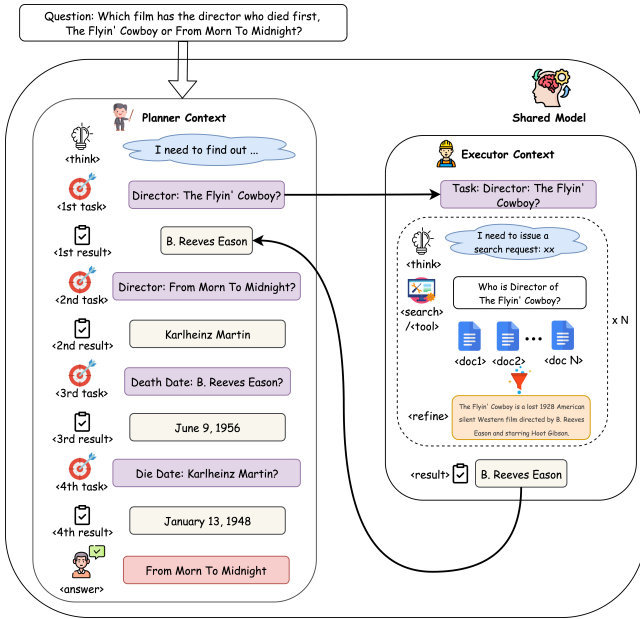


Figure 1: The workflow of CoDA. A single shared model acts as a Planner (left) to decompose the task and an Executor (right) to handle tool use in an isolated context.

- We achieve state-of-the-art performance on complex multi-hop QA benchmarks, as it significantly outperforms existing methods by as much as 6.0% in terms of accuracy, demonstrating its superior reasoning capabilities.
- We demonstrate the superior long-context robustness of our framework. Through targeted experiments, we show that unlike baseline models whose performance degrades sharply with increasing context length, CoDA maintains stable and high performance, directly validating its effectiveness in handling information-intensive tasks.

2 Related Work

2.1 Planner-Executor Agent Frameworks

To overcome the limitations of monolithic agents in long-horizon tasks, the field has increasingly adopted Planner-Executor frameworks. This paradigm separates high-level strategic planning from low-level action execution. Recent work explore this concept through diverse strategies for decomposition and learning.

Some frameworks, like CoAct [8] and PLAN-AND-ACT [4], decompose a complex goal into a sequence of natural language sub-tasks. Others, such as QCompiler [43], employ a more formal, neuro-symbolic approach, "compiling" a query into a structured execution plan (an Abstract Syntax Tree). Architecturally, designs range from modular, multi-agent systems like AgentOrchestra [42], which uses specialized agents for different sub-tasks, to unified models. Prior approaches often design the planner and executor separately, which leads to potential mismatches between the two components and cannot be optimized as a unified system. As a result, executors may fail to follow the planner's plans effectively, undermining overall task performance.

2.2 Retrieval-Augmented Reasoning Agents

Retrieval-Augmented Generation (RAG) has become a foundational technique for grounding Large Language Models (LLMs) in external, verifiable knowledge, thereby enhancing their factual accuracy and reducing hallucinations [1, 15]. The evolution of this paradigm has evolved from simple, single-step retrieval mechanisms [5, 10] to more sophisticated, iterative pipelines designed to handle complex queries. These advanced RAG systems incorporate techniques such as query decomposition [43], evidence refinement [39], and multi-round search [28, 35].

Despite these advances, traditional RAG methods often follow predefined workflows. This inherent rigidity can limit their ability to reason adaptively and make decisions dynamically when faced with novel or highly complex problems. To address this, a new class of agentic systems has emerged, integrating retrieval directly into the reasoning process as a learnable action [2, 9, 32]. In these frameworks, the agent can autonomously decide when and what to search based on its intermediate reasoning steps.

2.3 Context Engineering

The effective performance of Large Language Model (LLM) agents is critically dependent on the quality and structure of the information they process. This has led to the formalization of Context Engineering [19], which encompasses the systematic optimization of the LLM's informational payload, moving beyond simple prompt design. Our work is situated within this landscape, building upon foundational techniques and system-level architectures.

Research has identified the "lost-in-the-middle" problem, where models struggle to utilize information buried in long contexts [17]. In response, MemGPT [21] pioneered OS-inspired memory management, creating a hierarchy between a limited active context (main memory) and external storage, which aligns with our principle of context separation. These components are increasingly integrated into multi-agent Systems (MAS), a paradigm that directly informs our hierarchical, role-based framework. Systems like AutoGen [38] and MetaGPT [7] demonstrate how multiple agents, each with specialized roles and contexts, can collaborate to solve complex tasks. This approach embodies the principles of task specialization and context isolation, where each agent focuses on its own sub-problem.

2.4 RL for Agents

The paradigm for enhancing the capabilities of LLMs as agents is increasingly shifting from supervised fine-tuning (SFT) on curated datasets [23, 26], towards the more scalable and generalizable framework of RL. End-to-end RL offers a promising alternative to unlock the inherent reasoning capabilities of LLMs, with recent breakthroughs demonstrating its effectiveness [3]. The development of RL methods has progressed from foundational techniques like Reinforcement Learning from Human Feedback (RLHF) [13] and Proximal Policy Optimization (PPO) [27] to more advanced approaches such as Direct Preference Optimization (DPO) [25], and Group Relative Policy Optimization (GRPO) [29]. Among these methods, GRPO [29] is particularly relevant as it is specifically designed for LLMs. By replacing the traditional critic with a group-based evaluation strategy, GRPO has demonstrated strong performance in enhancing reasoning across tasks. In the domain of external

knowledge retrieval, systems such as Search-R1 [9], ReSearch [2], and R1-Searcher [32, 33] have successfully leveraged RL to enable models to autonomously develop reasoning strategies during the retrieval process, moving beyond manually specified cues. Building upon this proven efficacy and adaptability, our work extends GRPO to enhance retrieval-augmented reasoning, significantly improving the ability of LLMs to interact with external tools in question answering (QA) tasks. Furthermore, we further extend GRPO within a hierarchical agent framework, addressing the unique challenges of multi-step reasoning scenarios.

3 Method

In this section, we first formalize the problem and core challenges, and then introduce our proposed CoDA framework to address these challenges. Finally, we describe the RL strategy employed to train this framework, detailing our context-dependent policy updates and composite reward function.

3.1 Problem Formulation

3.1.1 Context Entanglement in Monolithic Agents. We address the task of complex, multi-step question answering where an agent, parameterized by a Large Language Model π_θ , interacts with a set of external tools T to answer a query Q . A common approach models this as a sequential decision-making process, where the agent generates a single, continuous trajectory $\tau = (y_1, \dots, y_L)$.

In this standard paradigm, the context C_t provided to the policy $\pi_\theta(y_t | C_t)$ at each step t is a monolithic concatenation of all prior history: $C_t = [Q; y_1; \dots; y_{t-1}]$. As the reasoning chain lengthens with multiple tool calls, this context rapidly accumulates raw, verbose documents (D_1, \dots, D_k) , leading to two widely recognized issues: computational inefficiency due to processing long sequences, and performance degradation as critical information gets “lost in the middle” [17] of an overwhelmingly long input.

We argue that these issues are symptoms of a deeper, structural flaw inherent in the monolithic design. Complex reasoning naturally comprises two distinct cognitive levels: high-level strategic planning (decomposing the problem) and low-level tactical execution (fulfilling a specific sub-task). This paradigm forces these two roles to operate within the same context, creating a state of *Context Entanglement*, which manifests as two primary challenges:

- **Strategic Context Pollution:** The Planner’s decision space, which should ideally focus on high-level goals and results, is contaminated by the noisy, low-level details of raw documents. This obscures the strategic signals required for effective long-range planning.
- **Execution Context Redundancy:** When the agent acts as an Executor for a specific task t , it is burdened with a context C_t containing the entire, often irrelevant, history of previous tasks $(\text{task}_1, \dots, \text{task}_{t-1})$. This not only dramatically inflates the context length, consuming valuable tokens and increasing latency, but also introduces unrelated information that can interfere with the focused execution of the current task.

Formally, the agent’s objective is to find the optimal parameters θ^* that maximize the expected cumulative reward $R(\tau)$ over

trajectories sampled from the policy:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] \quad (1)$$

However, due to the dual challenges of strategic pollution and context redundancy stemming from cognitive entanglement, the policy’s effectiveness $\pi_\theta(y_t | C_t)$ diminishes as task complexity and context length $|C_t|$ grow.

A seemingly intuitive approach to mitigate this context explosion would be to programmatically remove or truncate verbose tool outputs (e.g., the documents D_k) from the history C_t after they have been processed. However, naively truncating tool outputs at inference can introduce a distribution shift if the model was never trained under matching context dynamics. This introduces a severe **Training-Inference Skew**: the model is trained to generate each token conditioned on the full, unaltered history, but during inference, it would be forced to operate on an artificially shortened context it has never encountered during training. This distribution shift cripples the model’s ability to reason coherently, as its learned policy is no longer valid in this modified state space. This demonstrates that a simple post-hoc context truncation is not a viable solution. The problem of context entanglement requires a more principled, architectural redesign. Our work seeks to resolve this by fundamentally redesigning the agent’s operational framework to decouple these entangled roles.

3.2 Hierarchical Framework

To address the challenge, we propose the CoDA framework. Its core idea is to decompose the monolithic decision-making process into two synergistic yet contextually-isolated logical roles: a **Planner** and an **Executor**. Both roles are embodied by a **single** model π_θ but operate under distinct contexts.

3.2.1 Role Definitions and Context Spaces.

The Planner acts as a high-level strategist, operating within a concise **Strategic Context** C_P . Its responsibility is to generate sub-tasks or the final answer. The strategic context is kept clean, containing only high-level information:

$$C_P^{(t)} = \{Q, (\text{task}_1, \text{result}_1), \dots, (\text{task}_{t-1}, \text{result}_{t-1})\}$$

At each step t , the Planner generates the next sub-task, $\text{task}_t = \pi_\theta(C_P^{(t)})$. The process terminates when the Planner determines that it has gathered enough information. At this point, it generates a final action, which is to synthesize the collected results into the final answer. This final synthesis can be viewed as the last “task” in the sequence.

The Executor acts as a focused task handler, operating within a **Temporary Execution Context** C_E . It receives a single task t from the Planner and is responsible for its fulfillment through tool use and information processing. Its internal workflow is as follows:

- (1) **Initialize:** $C_E \leftarrow \{\text{task}_t\}$.
- (2) **Execute:** Invoke tools (e.g., $\mathcal{T}_{\text{search}}$) to retrieve raw documents D_t .
- (3) **Refine:** Summarize and distill D_t to produce a concise information summary $s_t = \text{Refine}(D_t)$.
- (4) **Conclude:** Generate a final, condensed result r_t based on the refined information within C_E . This result r_t is a

concise summary of the findings for task_t, ready to be passed back to the Planner.

Crucially, the verbose, raw documents D_t never enter the Planner’s strategic context C_p . The Executor functions as a black box that abstracts away noisy details, enabling effective context decoupling.

3.3 PECO (Planner-Executor Co-Optimization)

We train the CoDA framework end-to-end using an outcome-supervised reinforcement learning approach based on the GRPO algorithm [29]. Our core strategy hinges on applying a single trajectory-level reward to update the shared policy model π_θ , while ensuring the gradient for each token is calculated based on its specific role-dependent context (Planner or Executor).

3.3.1 Hierarchical Trajectory Generation (Rollout). Training begins by generating interaction trajectories using the behavior policy $\pi_{\theta_{\text{old}}}$. The standard rollout is illustrated as follows:

- (1) **Planner Action:** Starting with the user query, the Planner generates a sequence of thoughts and an action, which is either a <task> to delegate or a final <answer>.
- (2) **Executor Sub-Loop:** If the Planner generates a <task>, we invoke a separate generation loop. This session operates in an ephemeral context, initialized only with the sub-task, cutting the session off from the Planner’s long history. Within this ephemeral context, the model, now acting as the Executor, focuses solely on the sub-task. It can iteratively generate <search> actions, receive external tool outputs (e.g., <documents> . . . </documents>), and reason over them until it produces an <answer> for the sub-task.
- (3) **Returning to Planner:** The Executor’s final answer is packaged as a <result> and returned to the Planner, becoming part of the Planner’s strategic context for its next decision-making step.
- (4) **Trajectory Collection:** All generated sequences—the main Planner trajectory and all associated Executor trajectories—are collected. They are then concatenated into a single training batch, where each trajectory is explicitly labeled as either ‘planner’ or ‘executor’ to facilitate group-level credit assignment. This collection of trajectories, representing a complete multi-level reasoning process for a single query, forms a **trajectory group**.

3.3.2 Group-Level Credit Assignment. For each query, we generate $k > 1$ independent trajectories, termed as ‘rollouts’, using the same policy. All trajectories (Planner and Executor) originating from the same query form a trajectory group \mathcal{G} . We compute a scalar reward $R(\mathcal{G})$ for each \mathcal{G} based on the function detailed in Section 3.3.4.

Following GRPO[29] principles, we assign credit by normalizing the reward of each group against the statistics of its peers that tackle the same initial query. If the k rollouts for a single query yield a set of rewards $\{R(\mathcal{G}_1), R(\mathcal{G}_2), \dots, R(\mathcal{G}_k)\}$, the advantage $\hat{A}(\mathcal{G}_i)$ for the i -th group is calculated as:

$$\hat{A}(\mathcal{G}_i) = \frac{R(\mathcal{G}_i) - \text{mean}(R(\mathcal{G}_j)_{j=1}^k)}{\sigma(R(\mathcal{G}_j)_{j=1}^k)} \quad (2)$$

where mean and σ are the mean and std computed exclusively over the k outcomes for that specific query. This advantage is applied uniformly to every policy-generated token across all trajectories (both Planner and Executor) within that group \mathcal{G}_i . This outcome-based approach ensures that even intermediate Executor actions are credited based on their contribution to the Planner’s final success.

3.3.3 Context-Dependent Policy Update with Loss Masking. A cornerstone of our training strategy is ensuring the model learns exclusively from its own **actions**, not from **observations** provided by the environment. While the advantage signal $\hat{A}(\mathcal{G}_i)$ is uniform, the policy update is performed token-by-token, strictly enforcing this principle through loss masking.

We use the Group Relative Policy Optimization (GRPO) objective to update the policy π_θ against the behavior policy $\pi_{\theta_{\text{old}}}$. For each token y_t generated by π_θ within a group \mathcal{G}_i , the loss is:

$$L_{\text{CLIP}}(y_t, C_t, \theta) = \min \left(\rho_t(\theta) \hat{A}(\mathcal{G}_i), \text{clip}(\rho_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}(\mathcal{G}_i) \right) \quad (3)$$

where $\rho_t(\theta) = \frac{\pi_\theta(y_t|C_t)}{\pi_{\theta_{\text{old}}}(y_t|C_t)}$ is the importance sampling ratio.

The overall learning objective for the policy π_θ is to maximize the GRPO surrogate objective function $J(\theta)$, regularized by a KL divergence term against a reference policy $\pi_{\theta_{\text{ref}}}$:

$$J(\theta) = \mathbb{E}_{\mathcal{G} \sim \pi_{\theta_{\text{old}}}} \left[\sum_{\tau \in \mathcal{G}} \sum_{t=0}^{L_\tau-1} \left(L_{\text{CLIP}}(y_t, C_t, \theta) - \beta D_{\text{KL}}(\pi_\theta(y_t|C_t) \parallel \pi_{\theta_{\text{ref}}}(y_t|C_t)) \right) \cdot m_t \right]$$

The mask m_t is the crucial component that enforces the action-observation separation. During trajectory construction, our framework distinguishes between tokens generated by the agent and tokens returned by the environment. The loss is computed only on agent-generated tokens by applying a binary loss mask m_t :

- **Agent’s Actions ($m_t = 1$):** tokens are generated by the agent as part of its decision-making process. This includes the Planner’s reasoning and <task> generation, as well as the Executor’s reasoning and <search> generation.
- **Environmental Observations ($m_t = 0$):** tokens represent information from the environment. For the Planner, the output in the <result> . . . </result> block is an observation. For the Executor, the search result in the <documents> . . . </documents> block is also an observation.

3.3.4 Composite Reward Design. To guide the model towards mastering the complex behaviors required by our framework, we designed a composite reward function $R(\mathcal{G})$ that simultaneously encourages three distinct objectives: answer correctness, structural compliance, and information refinement quality, where total reward for a complete trajectory group \mathcal{G} is a weighted sum of them.

1. Answer Correctness (R_{ans}). The primary objective is to produce a factually correct answer. This reward is based on the F1 score between the final answer generated by the Planner A_{pred} , and the set of ground-truth answers G . We linearly transform this score to a range of $[-3, 3]$ to provide a strong, scaled reward signal.

$$S_{\text{ans}} = \max_{g \in G} \text{F1}(A_{\text{pred}}, g) \quad (4)$$

$$R_{\text{ans}}(\mathcal{G}) = 6 \cdot S_{\text{ans}} - 3 \quad (5)$$

2. *Format Compliance* (R_{format}). To ensure the agent adheres to the Plan-then-Execute protocol, we reward the generation of correctly formatted XML-style tags. This component is composed of two parts, one for the Planner and one for the Executor.

$$R_{\text{format}}(\mathcal{G}) = I_P(\mathcal{G}) + I_E(\mathcal{G}) \quad (6)$$

Here, $I_P(\mathcal{G})$ is an indicator function that equals 1 if the Planner’s final output in the group is structurally correct (i.e. proper use of `<task>` or `<answer>`). Similarly, $I_E(\mathcal{G})$ is an indicator function that equals 1 if all of the Executor’s outputs within the group are correctly formatted (i.e. proper use of `<search>` or `<result>`).

3. *Refinement Quality* (R_{refine}). A crucial capability of the Executor is to distill critical information from noisy search results. To incentivize not only the act of summarization but also its factual accuracy, the refinement reward is granted only if two conditions are met: a non-empty summary is produced, and this summary contains the ground-truth answer.

Let $s_{\text{refine}}(\tau_e)$ be the set of all string contents extracted from `<refine>` tags across all Executor trajectories within a group \mathcal{G} . We first concatenate these contents into a single, unified string, s_{combined} .

$$s_{\text{combined}} = \bigoplus_{i=1}^m s_i = s_1 \oplus s_2 \oplus \dots \oplus s_m \quad (7)$$

where \oplus denotes the string concatenation operator. The refine reward is formally defined as:

$$R_{\text{refine}}(\mathcal{G}) = \delta \cdot \mathbb{I}(s_{\text{combined}} \neq \emptyset) \cdot \mathbb{I}(G \subseteq s_{\text{combined}}) \quad (8)$$

where $\mathbb{I}(\cdot)$ is the indicator function, the condition $G \subseteq s_{\text{combined}}$ checks for the textual containment of at least one ground-truth answer from the set G within the combined refined string, and δ is its corresponding weight. This approach rewards the collective refined effort of the Executor, even if critical information is distributed across multiple refinement steps.

Total Reward. The final reward for a trajectory is the sum of these components:

$$R(\mathcal{G}) = R_{\text{ans}}(\mathcal{G}) + R_{\text{format}}(\mathcal{G}) + R_{\text{refine}}(\mathcal{G}) \quad (9)$$

This multi-faceted reward function effectively provides dense signals that guide the agent to concurrently optimize for accuracy, procedural correctness, and semantic understanding.

3.4 Analysis of Computational Complexity

To formally ground the computational advantages of CoDA, we analyze its time and space complexity during inference and contrast it with a standard monolithic agent architecture. Our analysis focuses on the peak context length, which dictates the worst-case computational cost. We start with the following notation.

- C : Context length
- $T(C) = \mathcal{O}(C^2)$: Time Complexity to generate one token.
- $S(C) = \mathcal{O}(C)$: Space Complexity.
- H : # of hops or sub-tasks in the Planner trajectory.
- h : # of reasoning hops in the Executor trajectory.
- L_{doc} : The average length of raw documents retrieved by a single search engine call. We assume L_{doc} is large.

- L_{res} : The average length of a concise result produced by the Executor. By design, $L_{\text{res}} \ll L_{\text{doc}}$.
- L_Q, L_{think} : The lengths of the initial query, intermediate thought traces and the issued task respectively. For asymptotic analysis, these are treated as lower-order terms.

3.4.1 Complexity of a Monolithic Agent. A monolithic agent operates in a single, cumulative context.

Definition 3.1 (Context Length of a Monolithic Agent). At step k ($1 \leq k \leq H$), the context length $C_{\text{mono}}(k)$ is the sum of all prior interactions. It can be approximated as:

$$C_{\text{mono}}(k) \approx L_Q + (k - 1) \cdot (L_{\text{think}} + L_{\text{doc}}) \quad (10)$$

The peak context length, $C_{\text{mono}}^{\text{max}}$, is reached at the final step H :

$$C_{\text{mono}}^{\text{max}} = C_{\text{mono}}(H) \approx H \cdot L_{\text{doc}} \quad (11)$$

PROPOSITION 3.2 (ASYMPTOTIC COMPLEXITY OF A MONOLITHIC AGENT). The space and time complexity for a monolithic agent are:

- Space: $S_{\text{mono}}(H) = \mathcal{O}(C_{\text{mono}}^{\text{max}}) = \mathcal{O}(H \cdot L_{\text{doc}})$
- Time: $T_{\text{mono}}(H) = \mathcal{O}((C_{\text{mono}}^{\text{max}})^2) = \mathcal{O}((H \cdot L_{\text{doc}})^2)$

Remark 3.1. The multiplicative term $H \cdot L_{\text{doc}}$ mathematically formalizes the "context explosion" problem, leading to intractable computational growth in multi-hop scenarios.

3.4.2 Executor Complexity. The Executor operates in an ephemeral workspace, which is re-initialized for each sub-task and does not accumulate history across hops.

Definition 3.3 (Executor Context Length). The peak context length within any Executor session, $C_{\text{exec}}^{\text{max}}$, is bounded by the size of the single raw document it processes:

$$C_{\text{exec}}^{\text{max}} \approx L_{\text{task}} + (h - 1) \cdot (L_{\text{think}} + L_{\text{doc}}) \approx \mathcal{O}(h \cdot L_{\text{doc}}) \quad (12)$$

Remark 3.2. $C_{\text{exec}}^{\text{max}}$ is independent of the total number of hops H .

3.4.3 Planner Complexity. The Planner maintains a strategic context, accumulating only the concise summaries from the Executor.

Definition 3.4 (Planner Context Length). At step k , the Planner’s context length $C_{\text{plan}}(k)$ is:

$$C_{\text{plan}}(k) \approx L_Q + (k - 1) \cdot (L_{\text{think}} + L_{\text{res}}) \quad (13)$$

The peak context length, $C_{\text{plan}}^{\text{max}}$, is reached at step H :

$$C_{\text{plan}}^{\text{max}} = C_{\text{plan}}(H) \approx H \cdot L_{\text{res}} \quad (14)$$

3.4.4 Overall CoDA Complexity. The system’s peak complexity is determined by the maximum context handled at any point, which is the greater of the Planner’s and any Executor’s peak context.

THEOREM 3.5 (ASYMPTOTIC COMPLEXITY OF CoDA). The peak context length for the CoDA framework is $C_{\text{CoDA}}^{\text{max}} = \max(C_{\text{plan}}^{\text{max}}, C_{\text{exec}}^{\text{max}})$. Consequently, its peak time and space complexities are:

- Space: $S_{\text{CoDA}}(H) = \mathcal{O}(\max(H \cdot L_{\text{res}}, h \cdot L_{\text{doc}}))$
- Time: $T_{\text{CoDA}}(H) = \mathcal{O}(\max((H \cdot L_{\text{res}})^2, (h \cdot L_{\text{doc}})^2))$

3.4.5 Conclusion of Analysis. Table 1 summarizes the asymptotic complexity. The results formally demonstrate CoDA’s computational advantages. While a monolithic agent’s complexity grows multiplicatively with the number of hops and document length $O(H \cdot L_{\text{doc}})$, CoDA transforms this into a dependency on the maximum of two separate terms: the planner’s context growth and the executor’s single-session context ($O(\max(H \cdot L_{\text{res}}, h \cdot L_{\text{doc}}))$).

By design, the length of a summarized result is significantly smaller than the raw document ($L_{\text{res}} \ll L_{\text{doc}}$). Furthermore, since the Planner decomposes the main task into simpler sub-tasks, the number of internal steps for the Executor (h) is typically a small constant (often $h = 1$). Consequently, CoDA effectively mitigates the "context explosion" problem by isolating the impact of large documents (L_{doc}) to isolated sessions, preventing their accumulation in the strategic planning context.

Table 1: Asymptotic complexity comparison for H -hop task.

Complexity Metric	Monolithic	CoDA (Hierarchical)
Space Complexity	$O(H \cdot L_{\text{doc}})$	$O(\max(H \cdot L_{\text{res}}, h \cdot L_{\text{doc}}))$
Time Complexity	$O((H \cdot L_{\text{doc}})^2)$	$O(\max((H \cdot L_{\text{res}})^2, (h \cdot L_{\text{doc}})^2))$

4 Experiments

In this section, we conduct a series of experiments to evaluate the effectiveness of CoDA, and to answer the following research questions:

- (1) How does CoDA compare against state-of-the-art baselines on single-hop and multi-hop QA benchmarks?
- (2) What is the impact of hierarchical architecture and composite reward function in CoDA?
- (3) How robust CoDA in long-context scenarios?

4.1 Experimental Setup

4.1.1 Datasets. We use the following seven widely used QA datasets, covering both single-hop and multi-hop reasoning.

- **Single-Hop QA:** Natural Questions (NQ) [14], TriviaQA [11], and PopQA [18].
- **Multi-Hop QA:** HotpotQA [40], 2WikiMultiHopQA [6], Musique [34], and Bamboogle [22].

To simulate a real-world open-domain setting, we remove all provided context documents.

4.1.2 Evaluation Metrics. We evaluate model performance using three standard metrics:

- **Exact Match (EM):** % of predictions that exactly match one of the ground-truth answers after normalization (e.g., lowercasing, removing punctuation and articles).
- **F1-Score:** The harmonic mean of precision and recall computed at the token level, which measures the overlap between the prediction and ground-truth answers.
- **CEM (Coverage Exact Match):** A more tolerant version of EM that scores 1 if a normalized ground-truth answer is a substring of the normalized prediction.

4.1.3 Baselines. We compare our CoDA against a range of strong baselines, including both closed-source and open-source models.

- **Closed-Source Models:** We include results from GPT-4o and GPT-4o-mini [20], representing state-of-the-art proprietary language models.
- **Open-Source Methods:** We evaluate several reasoning and search-augmentation methods built upon open-source LLMs. Most baseline results are sourced directly from Search-R1 [9], which shares our experimental setup. These methods include Naive Response, ReAct [41], Search-o1, Search-R1, and AutoRefine [31]. All open-source methods are evaluated with the Qwen2.5-3B-Base backbone for fairness.

4.1.4 Implementation Details. In all experiments, models perform search over the December 2018 Wikipedia dump [12], which serves as the external knowledge corpus. We use E5-base-v2 [37] as our retriever. For each search query, the retriever returns the top-3 most relevant documents. Our reinforcement learning (RL) framework is implemented using verl [30]. We employ Group Relative Policy Optimization (GRPO) [29] as the learning algorithm to train our agent, which is based on Qwen2.5-3B-Base.

4.2 Main Results (RQ1)

Table 2 presents the main results of our CoDA framework compared against a range of baseline methods on seven diverse QA benchmarks. Our proposed model, CoDA-Base, demonstrates state-of-the-art performance, achieving the highest average score of 0.407 across all datasets.

On single-hop QA tasks (NQ, TriviaQA, PopQA), which require less reasoning depth, CoDA performs only marginally better than the strong AutoRefine baseline, improving by approximately 2% on average. This is expected, as these tasks do not extensively challenge the model’s context management capabilities, and thus the primary advantage of our framework is less pronounced.

The true strength of the CoDA framework becomes evident in complex, multi-hop QA scenarios (HotpotQA, 2WikiMultiHopQA, Musique, Bamboogle). On these challenging benchmarks, CoDA consistently and significantly outperforms all baselines. The most notable gains are observed on 2WikiMultiHopQA and Musique—two notoriously difficult datasets requiring intricate reasoning over multiple documents. Here, CoDA surpasses the strong AutoRefine baseline by a remarkable 24% on 2WikiMultiHopQA and 10% on Musique.

This huge improvement on multi-hop tasks directly stems from CoDA’s design: hierarchical task decomposition and contextual separation. In monolithic agent architectures, the reasoning trace becomes progressively entangled with raw, unfiltered search results from multiple retrieval steps, where the model’s high-level plan is intermingled with low-level, noisy data. CoDA addresses this by separating the roles of the planner and executor, where the planner’s trajectory is shielded from raw search results.

4.3 Ablation Study (RQ2)

4.3.1 CoDA w/o Hierarchical Structure. To empirically validate that our hierarchical design is the primary driver of performance, we conducted an ablation study comparing agents with and without

Table 2: (RQ1) EM comparison of CoDA versus baseline methods with Qwen2.5-3B [24] across various QA benchmarks. Bold denotes best results; underline denotes the second-best; * * * indicates significance at the 0.01 level.

Methods	Single-Hop QA			Multi-Hop QA				Avg.
	NQ	TriviaQA	PopQA*	HotpotQA	2Wiki	Musique	Bamboogle	
w/o Retrieval								
Direct Generation	0.106	0.288	0.108	0.149	0.244	0.020	0.024	0.134
SFT	0.249	0.292	0.104	0.186	0.248	0.044	0.112	0.176
R1-Instruct [3]	0.210	0.449	0.171	0.208	0.275	0.060	0.192	0.224
R1-Base [3]	0.226	0.455	0.173	0.201	0.268	0.055	0.224	0.229
w/ Single-Hop Retrieval								
Naive RAG [15]	0.348	0.544	0.387	0.255	0.226	0.047	0.080	0.270
w/ Multi-Hop Retrieval								
Search-o1 [16]	0.238	0.472	0.262	0.221	0.218	0.054	0.320	0.255
IRCoT [36]	0.111	0.312	0.200	0.164	0.171	0.067	0.240	0.181
ReSearch-Base [2]	<u>0.427</u>	<u>0.597</u>	0.430	0.305	0.272	0.074	0.128	0.319
Search-R1-Base [9]	0.421	0.583	0.413	0.297	0.274	0.066	0.128	0.312
AutoRefine-Base [31]	0.424	0.587	<u>0.449</u>	<u>0.382</u>	<u>0.328</u>	<u>0.169</u>	<u>0.320</u>	<u>0.380</u>
CoDA-Base	0.441	0.608***	0.452	0.407***	0.408***	0.187	0.368	0.407***
Improve %	+0.01%	+3.58%	+0.67%	+6.54%	+24.39%	+10.65%	+15.00%	+7.11%

* PopQA contains duplicate queries with different gold answers, so we unify answers per query and re-evaluate.

Table 3: Ablation study comparing Search-R1 and CoDA with/without hierarchical structure. All values are EM scores.

Model Variants	Single-Hop QA			Multi-Hop QA				Avg.
	NQ	TriviaQA	PopQA	HotpotQA	2wiki	Musique	Bamboogle	
Search-R1								
w Hier.	0.441	0.601	0.423	0.388	0.284	0.151	0.339	0.375
w/o Hier.	0.421	0.583	0.413	0.297	0.274	0.066	0.128	0.312
CoDA								
w Hier.	0.441	0.608	0.452	0.407	0.408	0.187	0.368	0.407
w/o Hier.	0.424	0.587	0.449	0.382	0.328	0.169	0.320	0.380

this structure. We compared monolithic version of CoDA, (i.e. AutoRefine baseline). Additionally, to demonstrate the generalizability of the architectural benefit, we applied our hierarchical structure to the Search-R1 baseline. Table 3 provides compelling evidence for the superiority of our design. Applying the structure to the standard Search-R1 model boosts its average EM score from 0.312 to 0.375, with the most dramatic gains on complex multi-hop datasets like Musique. Similarly, our full CoDA model significantly outperforms its monolithic counterpart (0.407 vs. 0.380).

4.3.2 Planner & Executor Co-optimization(PECO). To isolate and validate the contribution of our reinforcement learning methodology, we conducted an ablation study comparing the full CoDA framework against a version that relies solely on prompting for its hierarchical behavior. We established a strong workflow baseline where the agent utilizes the same Planner-Executor architecture, but its actions are guided exclusively by the same carefully designed in-context prompts, with no RL-based parameter updates.

As shown in Figure 2, the results clearly demonstrate the critical role of our PECO training. On average, our full PECO-trained model achieves an F1-score of 0.50, a substantial 8 percentage point improvement over the prompt-guided baseline’s score of 0.42.

This performance gap is particularly evident on complex multi-hop QA datasets. For instance, on 2WikiMultiHopQA, PECO boosts the F1-score from approximately 0.34 to 0.48. Similarly, on Musique, the score improves from 0.20 to 0.29. This indicates that while the hierarchical structure provides a solid foundation, it is the reinforcement learning process that enables the agent to master the

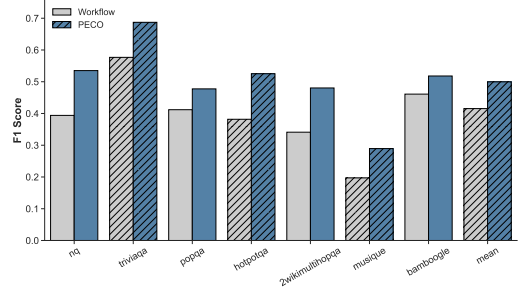


Figure 2: Ablation study comparing planner-executor co-optimization and a simple workflow.



Figure 3: Performance comparison over training steps.

Table 4: Ablation study over different reward contributions in CoDA. (F1)

Model Variants	Single-Hop QA			Multi-Hop QA				Avg.
	NQ	TriviaQA	PopQA	HotpotQA	2wiki	Musique	Bamboogle	
CoDA-Base	0.535	0.688	0.496	0.525	0.480	0.289	0.518	0.500
w/o Refine Reward	0.505	0.662	0.501	0.481	0.373	0.215	0.427	0.452
w/o Format Reward & Refine	0.187	0.331	0.147	0.188	0.187	0.085	0.200	0.190

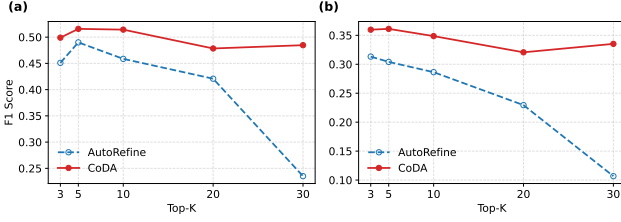
sophisticated strategies required for multi-step reasoning and evidence refinement. The experimental result underscores that static prompting is not enough. Through outcome-supervised RL, the agent learns nuanced and optimized policies for both planning and execution, developing a true coordination between the two roles that goes far beyond what prompt-following can achieve.

4.3.3 Reward Function Ablation. To dissect the contribution of each component in our composite reward function (Section 3.3.4), we conducted a detailed ablation study, shown in Figure 3 and Table 4.

Training the agent with only the final answer reward resulted in severe reward hacking; the agent would often answer directly bypassing the search tool and leaving the executor idle, leading to low accuracy and performance stagnation. Introducing a format compliance reward yielded a significant improvement. This component proved crucial for enforcing the structured plan-then-execute protocol. Furthermore, inspired by AutoRefine [31], we added a refinement reward to incentivize effective information distillation. This final model achieved the highest performance, highlighting the importance of rewarding both correct procedures and information processing quality.

Table 5: Performance comparison between AutoRefine and CoDA on more than 3 hops questions (Top-3 retrieval)

Model	Metric	3, 4 hops			Improve %
		2wiki	Musique	Avg.	
Qwen2.5-3B-Base					
AutoRefine	EM	0.452	0.060	0.256	-
	F1	0.464	0.138	0.301	-
	CEM	0.452	0.078	0.265	-
CoDA	EM	0.516	0.106	0.311	+21.48%
	F1	0.523	0.197	0.360	+19.60%
	CEM	0.516	0.137	0.326	+23.02%

**Figure 4: Performance robustness (a) across all datasets (b) multi-hop(≥ 3) questions as context length (Top-K) increases.**

4.4 Effect Analysis

4.4.1 Long-Context Robustness Test (RQ3). To evaluate our framework’s resilience against “context explosion”, we conducted a robustness test simulating information overload. We systematically increased the volume of potentially relevant information by varying the search tool’s top-k parameter (3 to 30), returning 200-token document chunks (up to 4K–6K tokens per call).

Figure 4a demonstrates CoDA’s superior robustness. The performance of the AutoRefine baseline shows a significant and steady decline as the number of retrieved documents increases. Its F1-score falls from 0.49 to 0.24, a 52% relative decrease, highlighting its vulnerability to cognitive overload.

In striking contrast, CoDA’s performance remains remarkably stable across all k values. This resilience is a direct consequence of our context-decoupled design. Because its workspace is ephemeral and its output is a condensed summary, the information overload is effectively contained and resolved at a low level, preventing pollution of the critical high-level planning process. It also aligns with our complexity analysis in Section 3.4: the peak complexity of a monolithic agent scales as $O((H \cdot L_{\text{doc}})^2)$, while CoDA scales as $O(\max((H \cdot L_{\text{res}})^2, (h \cdot L_{\text{doc}})^2))$, avoiding quadratic growth in the number of hops H when $L_{\text{res}} \ll L_{\text{doc}}$ and h is small.

4.4.2 Multi-hop Long-Context Settings. The challenge of context overload becomes particularly severe in multi-hop QA, where information from sequential retrieval steps accumulates and amplifies noise. We evaluate both CoDA and monolithic baselines on a set of 3,916 artificially selected questions requiring more than three reasoning hops, drawn from 2WikiMultiHopQA and Musique. As shown in Table 5, our method performs significantly better than AutoRefine, achieving average relative improvements of +21.48% in EM, +19.60% in F1, and +23.02% in CEM across the two multi-hop

datasets (3–4 hops), thereby demonstrating the effectiveness of the hierarchical structure in tackling complex multi-hop questions.

Figure 4b shows that the performance of monolithic agents like AutoRefine [31] exhibits a dramatic performance drop, as their context is increasingly polluted by raw documents retrieved at each hop. In contrast, CoDA demonstrates remarkable resilience. Its modular architecture effectively quarantines the noisy, long-context documents within the Executor’s ephemeral workspace. The *Executor* processes raw documents in an isolated workspace and outputs only concise summaries, while the *Planner* maintains a clean, high-level strategic context that is never directly exposed to these unprocessed inputs. This decoupling prevents information dilution and preserves reasoning quality. As a result, CoDA maintains stable and robust performance even when faced with extensive context on top of multi-hop reasoning challenges, demonstrating clear architectural advantages in complex, long-context, multi-hop scenarios where conventional approaches fail.

4.5 Case Study

As shown in Table 6, we present a case study on a complex multi-hop question from Musique, which requires: (1) identifying a film’s production company, (2) determining the city of its headquarters, and (3) finding the Greyhound bus station in that city. AutoRefine issues a single broad query covering the entire task, returning noisy and ambiguous results. This leads to an early hallucination—incorrectly identifying *Sony Pictures Animation* as the production company—which sends the agent down a wrong reasoning path and yields an incorrect final answer.

In contrast, CoDA’s hierarchical structure demonstrates superior robustness. The Planner methodically decomposes the problem into three distinct, manageable sub-tasks: (1) identify the production company, (2) find its headquarters’ city, and (3) find the bus station in that city. Each sub-task is delegated to an Executor sequentially, which operates in a clean, isolated context. As shown in the Executor’s trace, this isolation allows it to formulate a simple, precise search query (“production company that produced A Cosmic Christmas”), which yields the correct entity, “Nelvana”. It clearly shows that the accumulation of unfiltered, verbose information in a single context window pollutes the reasoning process, making the monolithic agent prone to early-stage errors, thus leading to final failure. However, CoDA’s design principle of context decoupling directly leads to reliable task-decomposition and step-by-step reasoning.

5 Conclusion

In this paper, we presented CoDA, a backbone-shared Planner–Executor framework that decouples high-level planning from low-level tool use, trained end-to-end using an outcome-supervised RL scheme. It achieves significant performance improvements over state-of-the-art baselines across seven QA benchmarks, demonstrating the superiority of our design. As our findings primarily hold for open-domain multi-hop QA with Qwen2.5-3B under matched retrieval settings, we plan to further explore its effectiveness in larger open-source models and different task families. We also plan to investigate nested CoDA for >5-hop problems and report accuracy and stability trade-offs.

Table 6: Case Study. The predictions are colored Red if the model answers incorrectly and Green if it answers correctly. The core evidences are colored Blue.

Question: Where do greyhound buses leave from the in the city where the headquarters of the production company which produced A Cosmic Christmas is located?

Ground Truth: **Toronto Coach Terminal**

AutoRefine:

<think> To answer the question "Where do greyhound buses leave from the city where the headquarters of the production company which produced A Cosmic Christmas is located?", I'll issue a search request.</think>

<search> Where do greyhound buses leave from the city where the headquarters of the production company which produced A Cosmic Christmas is located </search> <documents>[Doc 1: Williams Street]...[Doc 2: Transportation Manufacturing Corporation]...[Doc 3: Liverpool Santa Dash] ... [Doc 19: The Big Bus] [Doc 20: Atlantic Greyhound Lines]...</documents>

<refine> From the search results, I know that the production company which produced A Cosmic Christmas is Sony Pictures Animation. To find out the city where the headquarters of Sony Pictures is located, I'll issue another search request.</refine> c

<search> Where is the headquarters of Sony Pictures located </search> <documents>[Doc 1: Sony Pictures] Sony Pictures Sony Pictures Entertainment, Inc. ... Based in **Culver City, California**, it encompasses Sony's motion picture, television production and distribution. [Doc 2: Sony Pictures Studios]...[Doc 3: History of Sony] ... [Doc 19: Sony Pictures Studios] [Doc 20: Sony]...</documents>

<refine> From the search results, I know that the headquarters of Sony Pictures is located in Culver City, California. To find out where greyhound buses leave from the city where the headquarters of Sony Pictures is located, I'll issue another search request.</refine>

<search> Where do greyhound buses leave from the city Culver City, California</search> <documents> [Doc 1: Culver CityBus]... [Doc 10: Greyhound Lines](San Jose and Oakland). A stop in the city of San Francisco was added in December 2013 along with a new route between Los Angeles and Las Vegas. [Doc 20: Transportation in California]...</documents>

<refine> From the search results, I know that greyhound buses leave from Culver City, California.</refine> <answer> **Culver City** </answer>.

CoDA:

(Planner Trajectory)

<think> I need to find out where Greyhound buses leave from the city where the headquarters of the production company that produced A Cosmic Christmas is located. </think>

<task> What is the production company that produced A Cosmic Christmas? </task>

<result> **Nelvana** </result>

<task> What is the city where the headquarters of Nelvana is located? </task>

<result> **Toronto, Ontario** </result>

<task> Where do Greyhound buses leave from Toronto, Ontario? </task>

<result> **Toronto Coach Terminal**</result>

<answer> **Toronto Coach Terminal** </answer>

(Executor Trajectory)

Executor 1 <task> What is the production company that produced A Cosmic Christmas? </task>

<think> I need to find out the production company that produced A Cosmic Christmas. </think>

<search> production company that produced A Cosmic Christmas </search> <documents>[Doc 1: A Cosmic Christmas] A Cosmic Christmas A Cosmic Christmas is one of the first productions made by **Nelvana**. It premiered on December 4, 1977, in Canada on CBC Television. ... [Doc 2: A Cosmic Christmas]...[Doc 3: Christmas on Mars] ... [Doc 19: A Cosmic Christmas] [Doc 20: Cosmos: A Spacetime Odyssey]...</documents>

<refine> Based on the documents, the production company that produced A Cosmic Christmas is Nelvana. </refine>

<result> **Nelvana** </result>

...

Executor 3 <task> Where do Greyhound buses leave from Toronto, Ontario? </task>

<think> I need to find out where Greyhound buses leave from Toronto, Ontario. </think>

<search> Greyhound buses leaving from Toronto, Ontario </search> <documents> [Doc 1: A Greyhound Canada] [Doc 2: Greyhound Canada]... A list of cities served by this service: NeOn, ... is a service operated by Greyhound Canada in cooperation with Trailways of New York and Greyhound Lines between the New Yorker Hotel in Manhattan and **the Toronto Coach Terminal**.... [Doc 3: Bus companies in Ontario] ... [Doc 19: Bus companies in Ontario] [Doc 20: Greyhound Canada] </documents>

<refine> Based on the documents, Greyhound buses leave from the Toronto Coach Terminal in Toronto, Ontario.</refine>

<result> **Toronto Coach Terminal** </result>

References

- [1] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. arXiv:2112.04426 [cs.CL] <https://arxiv.org/abs/2112.04426>
- [2] Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025. ReSearch: Learning to Reason with Search for LLMs via Reinforcement Learning. arXiv:2503.19470 [cs.AI] <https://arxiv.org/abs/2503.19470>
- [3] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] <https://arxiv.org/abs/2501.12948>
- [4] Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-Act: Improving Planning of Agents for Long-Horizon Tasks. arXiv:2503.09572 [cs.CL] <https://arxiv.org/abs/2503.09572>
- [5] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL]

- <https://arxiv.org/abs/2312.10997>
- [6] 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*, Donia Scott, Nuria Bel, and Chengqing Zong (Eds.). International Committee on Computational Linguistics, Barcelona, Spain (Online), 6609–6625. doi:10.18653/v1/2020.coling-main.580
 - [7] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiaowu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI] <https://arxiv.org/abs/2308.00352>
 - [8] Xinming Hou, Mingming Yang, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Wayne Xin Zhao. 2024. CoAct: A Global-Local Hierarchy for Autonomous Agent Collaboration. arXiv:2406.13381 [cs.CL] <https://arxiv.org/abs/2406.13381>
 - [9] 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:2503.09516 [cs.CL] <https://arxiv.org/abs/2503.09516>
 - [10] Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. 2025. FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation Research. In *Companion Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) (WWW '25). Association for Computing Machinery, New York, NY, USA, 737–740. doi:10.1145/3701716.3715313
 - [11] Mandar Joshi, Eunsol Choi, Daniel 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)*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, Vancouver, Canada, 1601–1611. doi:10.18653/v1/P17-1147
 - [12] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 6769–6781. doi:10.18653/v1/2020.emnlp-main.550
 - [13] Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. 2024. A Survey of Reinforcement Learning from Human Feedback. arXiv:2312.14925 [cs.LG] <https://arxiv.org/abs/2312.14925>
 - [14] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466. doi:10.1162/tacl_a_00276
 - [15] 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. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL] <https://arxiv.org/abs/2005.11401>
 - [16] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic Search-Enhanced Large Reasoning Models. arXiv:2501.05366 [cs.AI] <https://arxiv.org/abs/2501.05366>
 - [17] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranajpe, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172 [cs.CL] <https://arxiv.org/abs/2307.03172>
 - [18] 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)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 9802–9822. doi:10.18653/v1/2023.acl-long.546
 - [19] Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. 2025. A Survey of Context Engineering for Large Language Models. arXiv:2507.13334 [cs.CL] <https://arxiv.org/abs/2507.13334>
 - [20] OpenAI. 2024. GPT-4o System Card. arXiv:2410.21276 [cs.CL] <https://arxiv.org/abs/2410.21276>
 - [21] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560 [cs.AI] <https://arxiv.org/abs/2310.08560>
 - [22] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and Narrowing the Compositionality Gap in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 5687–5711. doi:10.18653/v1/2023.findings-emnlp.378
 - [23] 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. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. arXiv:2307.16789 [cs.AI] <https://arxiv.org/abs/2307.16789>
 - [24] Qwen, ., An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuhong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] <https://arxiv.org/abs/2412.15115>
 - [25] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] <https://arxiv.org/abs/2305.18290>
 - [26] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL] <https://arxiv.org/abs/2302.04761>
 - [27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] <https://arxiv.org/abs/1707.06347>
 - [28] 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*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 9248–9274. doi:10.18653/v1/2023.findings-emnlp.620
 - [29] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeek-Math: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv:2402.03300 [cs.CL] <https://arxiv.org/abs/2402.03300>
 - [30] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. HybridFlow: A Flexible and Efficient RLHF Framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*. ACM, 1279–1297. doi:10.1145/3689031.3696075
 - [31] Yaorui Shi, Sihang Li, Chang Wu, Zhiyuan Liu, Junfeng Fang, Hengxing Cai, An Zhang, and Xiang Wang. 2025. Search and Refine During Think: Autonomous Retrieval-Augmented Reasoning of LLMs. arXiv:2505.11277 [cs.CL] <https://arxiv.org/abs/2505.11277>
 - [32] Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-Searcher: Incentivizing the Search Capability in LLMs via Reinforcement Learning. arXiv:2503.05592 [cs.AI] <https://arxiv.org/abs/2503.05592>
 - [33] Huatong Song, Jinhao Jiang, Wenqing Tian, Zhipeng Chen, Yuhuan Wu, Jiahao Zhao, Yingqian Min, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-Searcher++: Incentivizing the Dynamic Knowledge Acquisition of LLMs via Reinforcement Learning. arXiv:2505.17005 [cs.CL] <https://arxiv.org/abs/2505.17005>
 - [34] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics* 10 (2022), 539–554. doi:10.1162/tacl_a_00475
 - [35] 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)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 10014–10037. doi:10.18653/v1/2023.acl-long.557
 - [36] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. arXiv:2212.10509 [cs.CL] <https://arxiv.org/abs/2212.10509>
 - [37] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2024. Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv:2212.03533 [cs.CL] <https://arxiv.org/abs/2212.03533>
 - [38] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 [cs.AI] <https://arxiv.org/abs/2308.08155>
 - [39] Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation. arXiv:2310.04408 [cs.CL] <https://arxiv.org/abs/2310.04408>

- [40] 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 2369–2380. doi:10.18653/v1/D18-1259
- [41] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] <https://arxiv.org/abs/2210.03629>
- [42] Wentao Zhang, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. 2025. AgentOrchestra: A Hierarchical Multi-Agent Framework for General-Purpose Task Solving. arXiv:2506.12508 [cs.AI] <https://arxiv.org/abs/2506.12508>
- [43] Yuyao Zhang, Zhicheng Dou, Xiaoxi Li, Jiajie Jin, Yongkang Wu, Zhonghua Li, Qi Ye, and Ji-Rong Wen. 2025. Neuro-Symbolic Query Compiler. arXiv:2505.11932 [cs.CL] <https://arxiv.org/abs/2505.11932>