Topic:
Agentic Retrieval-Augmented Generation (RAG)

Presenters: Alexis Liao, Xuanzhen Lao, Feng Qiao

Adaptive-RAG:
Learning to Adapt Retrieval-Augmented
Large Language Models through Question Complexity

Presenter: Alexis Liao
Date: October 30, 2025

The Big Picture: RAG is Great, But...

- **Retrieval-Augmented Generation (RAG)** improves LLM accuracy by providing external knowledge, which is crucial for Question-Answering (QA) tasks
- However, a **"one-size-fits-all"** strategy is inefficient
- **The Problem:** Not all questions are created equal. Some are simple, while others are complex
- **The Challenge:** How can we be both **accurate** for complex questions and **efficient** for simple ones?

Current Approaches and Their Limits

Approach	How It Works	Weakness
(A)Single-Step	Retrieves documents once, then generates an answer.	**Inaccurate** for complex queries that need multiple pieces of information.
(B)Multi-Step	Iteratively retrieves documents and refines the answer.	**Inefficient** and slow for simple queries, wasting time and resources.

The Solution: Adaptive-RAG

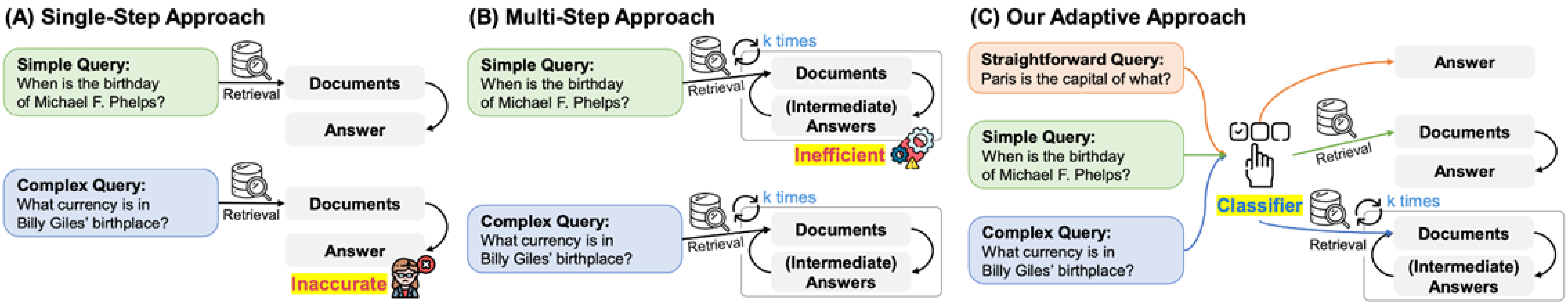
- This paper proposes a new framework: **Adaptive Retrieval-Augmented Generation (Adaptive-RAG)**
- **The Core Idea:** Don't use one strategy for all queries
  Instead, **dynamically select the best strategy** based on the question's complexity
- This creates a balanced system that is both **accurate and efficient**

How It Works: The Query Complexity Classifier

- Adaptive-RAG first sends every question to a **Classifier**
- This is a smaller, faster language model (T5-Large) trained to do one thing: judge the complexity of the question
- It sorts each question into one of three categories:
  - **A: Straightforward** -> Answerable by the LLM alone (No Retrieval)
  - **B: Simple** -> Needs one retrieval step (Single-step)
  - **C: Complex** -> Needs a multi-step, iterative approach

The Adaptive-RAG Workflow

- This diagram shows the complete workflow
- **(A) and (B)** are the inefficient, one-size-fits-all approaches
- **(C) is our solution.** The **Classifier** acts as a router, sending queries down the most efficient path based on their complexity

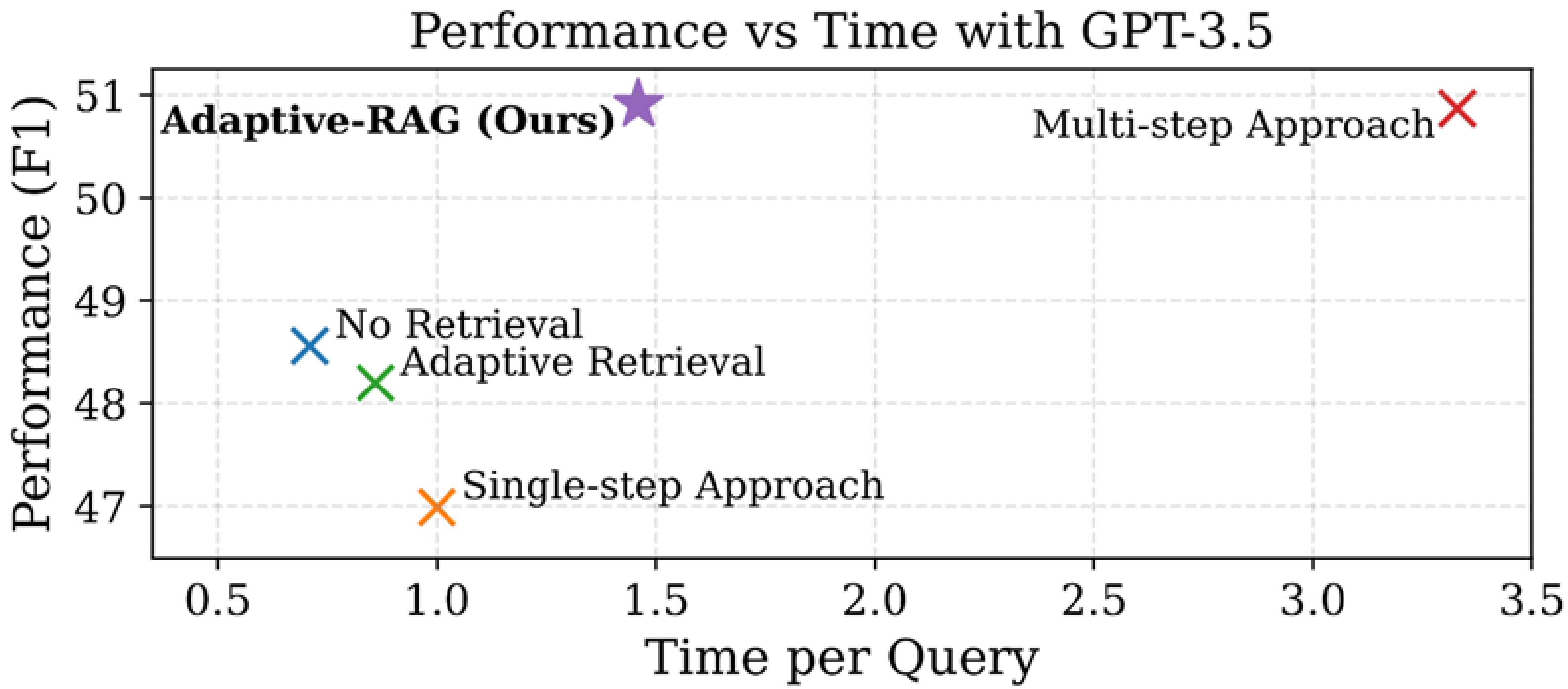


A Clever Training Strategy

- **Problem:** There's no dataset labeled with "query complexity"
- **Solution:** The authors created one automatically
- **How:**
    1. **Analyze Model Performance:** They ran thousands of questions through all three strategies (No, Single, and Multi-step). If the simplest model got it right, the question was labeled as simple.
    2. **Use Dataset Bias:** They used existing QA datasets. Questions from datasets known to have multi-step answers (like HotpotQA) were automatically labeled "Complex"

Overall Performance

- **Finding:** This chart shows Adaptive-RAG (the star) in the ideal top-left corner. It achieves the highest performance (F1 score), even beating the complex "Multi-step Approach," while being **more than twice as fast**



Main Results

- **Finding:** This table shows the main results across all models. Adaptive-RAG consistently achieves the best balance of accuracy (F1/EM) and efficiency (Time), proving it's not just a "one-time" success but a robust method

Table 1: Averaged results on a collection of benchmark datasets for open-domain question answering including the single-hop and multi-hop queries, with different LLMs. Self-RAG* is trained with a different base LLM, namely LLaMA2 (Touvron et al., 2023); therefore, we compare the results of FLAN-T5-XL (3B) with the results from Self-RAG with LLaMA2 (7B) and the results of others with the results from Self-RAG with LLaMA2 (13B). We emphasize our results in bold, for easy comparisons.

		FLAN-T5-XL (3B)					FLAN-T5-XXL (11B)					GPT-3.5 (Turbo)				
Types	**Methods**	EM	F1	Acc	Step	Time	EM	F1	Acc	Step	Time	EM	F1	Acc	Step	Time
Simple	**No Retrieval**	14.87	21.12	15.97	0.00	0.11	17.83	25.14	19.33	0.00	0.08	35.77	48.56	44.27	0.00	0.71
	Single-step Approach	34.83	44.31	38.87	1.00	1.00	37.87	47.63	41.90	1.00	1.00	34.73	46.99	45.27	1.00	1.00
Adaptive	**Adaptive Retrieval**	23.87	32.24	26.73	0.50	0.56	26.93	35.67	29.73	0.50	0.54	35.90	48.20	45.30	0.50	0.86
	Self-RAG*	9.90	20.79	31.57	0.72	0.43	10.87	22.98	34.13	0.74	0.23	10.87	22.98	34.13	0.74	1.50
	Adaptive-RAG (Ours)	**37.17**	**46.94**	**42.10**	**2.17**	**3.60**	**38.90**	**48.62**	**43.77**	**1.35**	**2.00**	**37.97**	**50.91**	**48.97**	**1.03**	**1.46**
Complex	**Multi-step Approach**	39.00	48.85	43.70	4.69	8.81	40.13	50.09	45.20	2.13	3.80	38.13	50.87	49.70	2.81	3.33
Oracle	**Adaptive-RAG w/ Oracle**	45.00	56.28	49.90	1.28	2.11	47.17	58.60	52.20	0.84	1.10	47.70	62.80	58.57	0.50	1.03

Classifier Performance

- **Finding:** These charts show our classifier is more accurate than other adaptive methods, which leads to better QA performance. The confusion matrix on the right shows us exactly where the classifier makes mistakes, like confusing "No" retrieval with "One" retrieval.

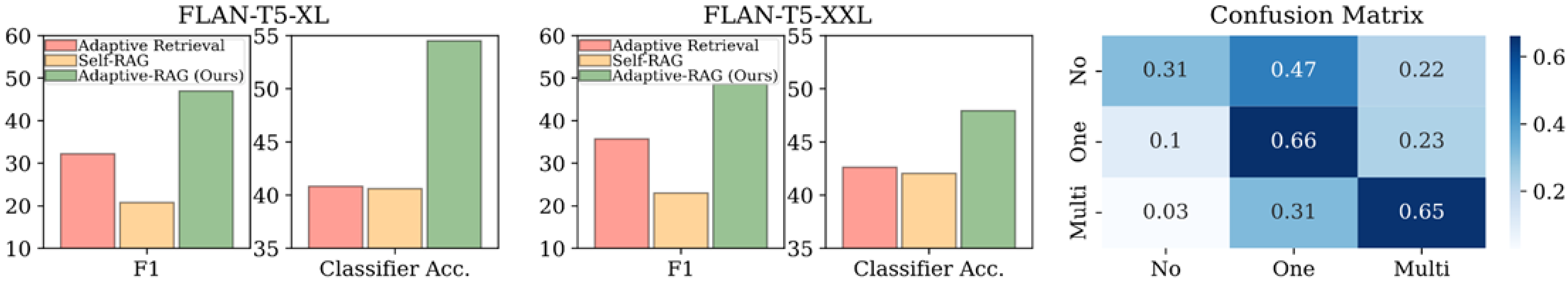


Figure 3: Performance on QA and query-complexity assessment of different adaptive approaches for retrieval-augmented LLMs with FLAN-T5 XL (Left) and XXL (Center). For labeling the complexity of queries, we use the silver data annotated from the prediction outcomes of models (described in Section 3.2). We also provide the confusion matrix across three labels (Right).

Case Study

Finding: This case study perfectly shows *why* the classifier is so important

- For the **simple question** (top), the baseline fails, but Adaptive-RAG correctly skips retrieval and gets the answer ("Google")
- For the **complex question** (bottom), the baseline fails again, but Adaptive-RAG correctly identifies it as complex and activates the multi-step process to find the correct answer ("Sebastian Cabot")

Table 5: Case study with GPT-3.5 (Turbo), where we represent the factual error in red and the accurate information in blue.

Dataset	Question	Adaptive Retrieval	Adaptive-RAG (Ours)
NQ (Single-hop)	Which famous corporate logo changed to a flat colour/color sans serif font in its first major change since 1999?	**Query Type:** B (Single-step Approach) **Reasoning:** The famous corporate logo that changed to a flat color sans-serif font in its first major change since 1999 is Microsoft. Microsoft changed its logo in 2012, replacing the italicized, multi-colored font with a simple, flat, grey font. So the answer is: Microsoft.	**Query Type:** A (Non Retrieval) **Reasoning:** The famous corporate logo that changed to a flat color sans serif font in its first major change since 1999 is Google. Google changed its logo in 2015. So the answer is: Google.
MuSiQue (Multi-hop)	Who is the child of the Italian navigator who explored the eastern coast of the continent César Gaytan was born in for the English?	**Query Type:** A (Non Retrieval) **Reasoning:** César Gaytan was born in the continent of North America. The Italian navigator who explored the eastern coast of North America for the English was Giovanni Caboto, also known as John Cabot. So the answer is: Giovanni Caboto/John Cabot.	**Query Type:** C (Multi-step Approach) **Reasoning:** Césarr Gaytan was born in Guadalajara, Jalisco, Mexico. The Italian navigator who explored the eastern coast of the continent for the English is John Cabot. John Cabot's son is Sebastian Cabot. So the answer is: Sebastian Cabot.

Conclusion & Key Takeaways

1. **The Problem is Real:** A single RAG strategy is not optimal for real-world use where query complexity varies.
2. **Adaptation is the Solution:** Adaptive-RAG successfully balances the trade-off between accuracy and efficiency.
3. **The Classifier is Key:** By quickly assessing a query's complexity, the system can route it to the most appropriate and cost-effective pathway.
4. **The Future is Efficient:** This approach points toward building smarter, more resource-aware AI systems.

Thank You

Questions?

Auto-rag: Autonomous Retrieval-augmented
Generation For Large Language Models

Tian Yu[1,3], Shaolei Zhang[1,3], Yang Feng[1,2,3] *∗
Presenter: Xuanzhen Lao

Motivation

- LLMs are knowledge-limited they forget or can’t access recent data.
- RAG enhances them by retrieving external knowledge.
- But traditional RAG often uses fixed, rule-based retrieval.

- ❓**Question**❓**:** Can the model decide when and what to retrieve by itself?

Background

- Auto-RAG enables LLMs to plan, refine, and stop retrieval autonomously — achieving efficient, adaptive information gathering.

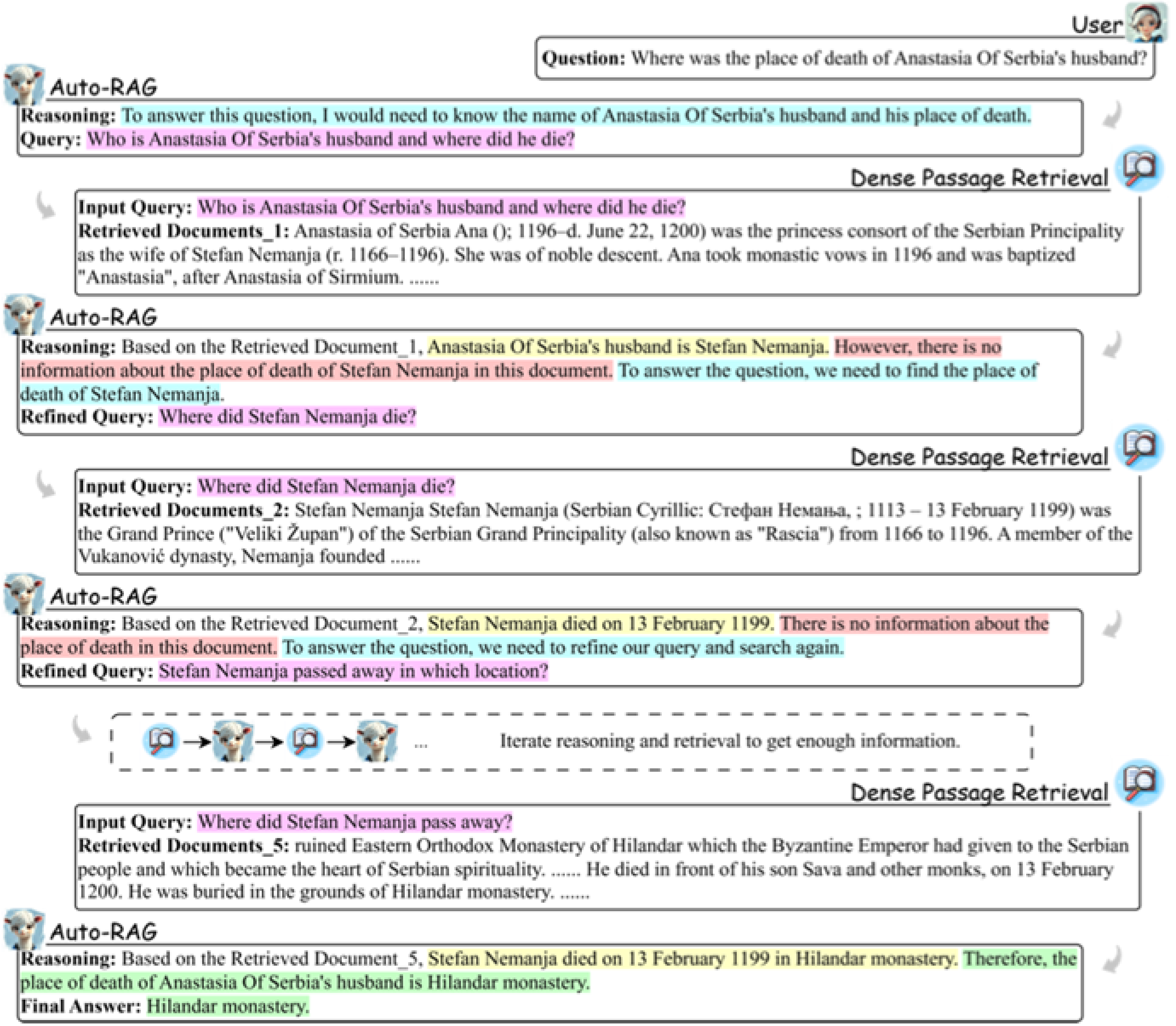


Figure 1: A concrete example of how Auto-RAG addresses complex multi-hop questions. Auto-RAG engages in iterative reasoning, strategically plans retrievals, extracts relevant knowledge, precisely identifies information needs, and refines query for the next retrieval, ultimately converging on the final answer. In this example, Auto-RAG terminates after five interactions with the retriever, successfully yielding the correct answer.

Key Features

- **Autonomous decision-making** : LLM decides *when* and *what* to retrieve.
- **Multi-turn refinement:** Iteratively improves query and retrieval quality.
- **Adaptive iteration:** Adjusts rounds by task complexity.
- **Interpretability:** Explains retrieval steps in natural language.

Core Mechanism: Autonomous Decision Loop

Three main decisions:

- *When to Retrieve* — trigger new retrieval if current knowledge insufficient.
- *What to Retrieve* — generate refined queries based on gaps.
- *When to Stop* — stop when the model believes the information is adequate.

These decisions are made by the LLM itself via instruction-based reasoning.

Method: Reasoning Based Planning And Query Refinement

Auto-RAG multiple autonomous steps:

1. **Retrieval Planning:** Analyze the user query and plan what to search.
2. **Query Execution:** Retrieve relevant documents from the knowledge base.
3. **Information Extraction:** After retrieval, the model extracts key info and decides if more queries are needed.
4. **Answer Generation:** Generate the final response based on all retrieved info.

Method- Data Construction for Training

The data synthesis involves:

1. Providing the LLM with initial user input to predict what knowledge is necessary (R0).
2. Generating a sequence of queries (Qgen) based on the input and previously retrieved documents (Rt-1).
3. Retrieving documents based on these queries and determining if they contain a sub-answer.
4. Refining queries based on the retrieved documents until the model can generate a final answer (A).

Training Procedure

- The model is fine-tuned with supervised learning using cross-entropy loss.
- Training data come from the synthetic reasoning–retrieval dataset.
- This helps the model learn when to retrieve, how to refine queries, and when to stop during inference.

$$\mathcal{L} = -\sum_{0 \le t \le T} \log \Pr(y_t | x_{\le t}, y_{<t}),$$

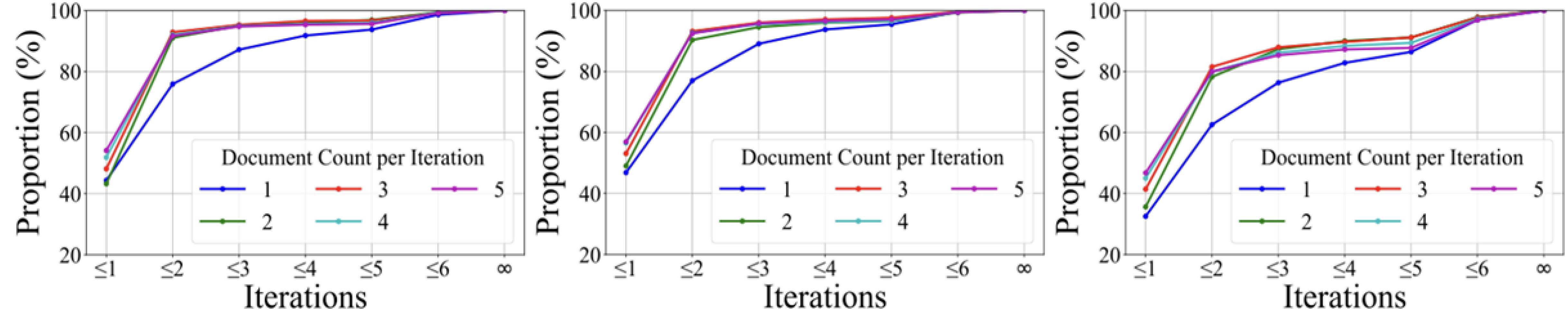
Main Results

- Auto-RAG achieves the highest average score , surpassing Self-RAG and FLARE.

Methods	NQ	2Wiki	TQA	PQA	HQA	WQ	AVG
	EM	F1	EM	F1	F1	EM	
No Retrieval							
Naive Gen	22.6	33.9	55.7	21.7	28.4	18.8	30.2
Single-time Retrieval							
Standard RAG	35.1	21.0	58.8	36.7	35.3	15.7	33.8
IRCoT	33.3	32.4	56.9	45.6	41.5	20.7	38.4
REPLUG	28.9	21.1	57.7	27.8	31.2	20.2	31.2
RECOMP-abstractive	33.1	32.4	56.4	39.9	37.5	20.2	36.6
Selective-Context	30.5	18.5	55.6	33.5	34.4	17.3	31.6
Iterative Retrieval							
FLARE	22.5	33.9	55.8	20.7	28.0	20.2	30.2
Self-RAG	36.4	25.1	38.2	32.7	29.6	21.9	30.7
Iter-RetGen	36.8	21.6	60.1	37.9	38.3	18.2	35.5
Ours (Autonomous Retrieval)							
Auto-RAG	**37.9**	**48.9**	**60.9**	**47.8**	**44.9**	**25.1**	$\boxed{\textbf{44.3}}$

Analysis 1: Strong Adaptability To Questions And Retrievers

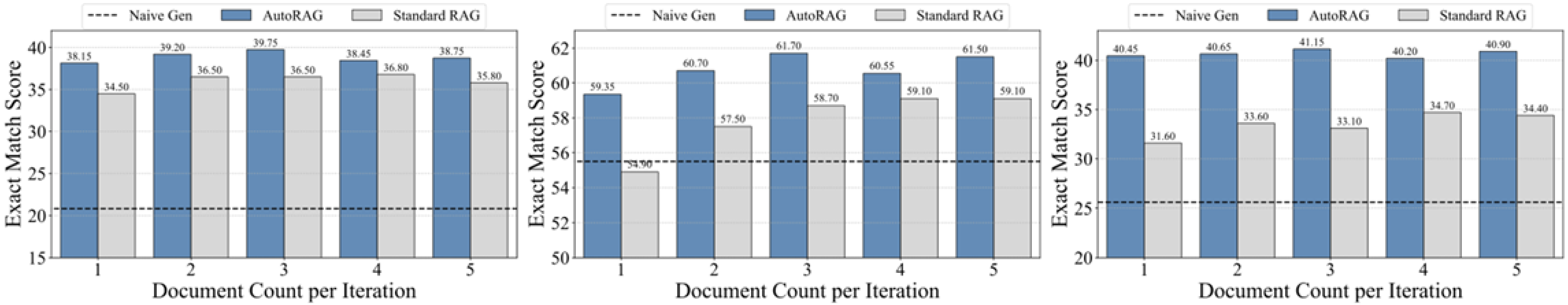
- Auto-RAG adapts iteration count to task complexity.
- Single-hop QA (NQ, TriviaQA): more one-round stops.
- Multi-hop QA (HotpotQA): more retrieval rounds.
- Auto-RAG can adjusts the retrieval depth based on inference difficulty.



(a) NQ (b) TriviaQA (c) HotpotQA

Analysis 2: Strong Adaptability To Questions And Retrievers

- More docs lead to earlier termination.
- Optimal at about 3 docs per iteration.
- Better than Standard RAG and Naive Gen.



(a) NQ (b) TriviaQA (c) HotpotQA

Figure 4: QA performance of Auto-RAG with varying document counts provided per iteration.

Ablation Study

The impact of training, reasoning, and zero-shot refinement on performance was verified.

•w/o training: big drop → fine-tuning essential.

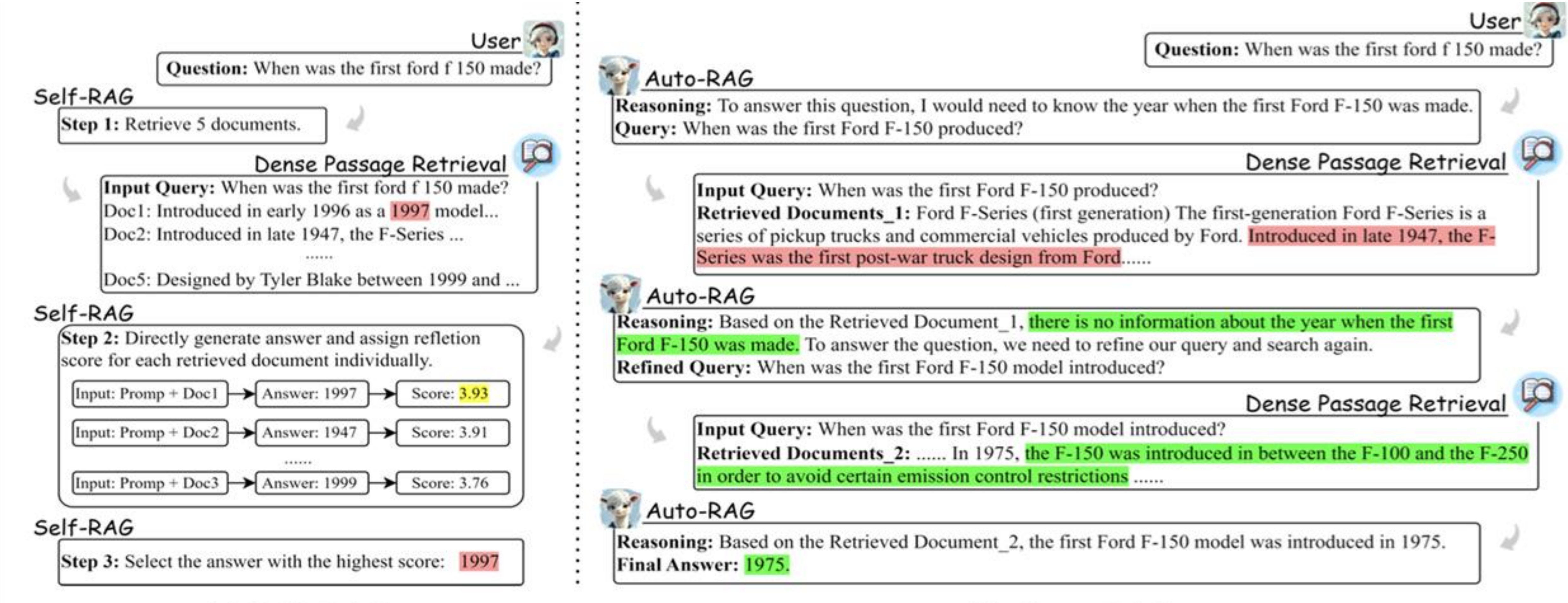
•w/o reasoning: largest drop → reasoning core factor.

•w/o zero-shot: slight drop → diverse queries help.

Methods	**NQ** EM	**2Wiki** F1	**TQA** EM	**PQA** F1	**HQA** F1	**WQ** EM	**AVG**
AutoRAG	**37.9**	**48.9**	**60.9**	**47.8**	**44.9**	**25.1**	**44.3**
w/o training	32.7	39.5	56.4	42.7	40.3	19.1	38.5
w/o reasoning	31.9	26.6	55.6	44.2	36.0	17.6	35.3
w/o zero-shot refinement	36.8	44.0	60.2	45.1	42.9	22.2	41.9

Case Study

- **Self-RAG:** only selects among fixed results using reflection tokens
- **Auto-RAG**：autonomously decides when and what to retrieve, continuing the search until relevant information is found.



(a) Self-RAG (b) Auto-RAG

Limitations

- Extra computation cost (multi-round retrieval)
- Depends on retriever & knowledge base quality
- Tested only on text tasks (no multimodal extension yet)

Discussion

- Auto-RAG makes LLMs autonomous in query refinement & retrieval
- Uses reasoning to decide what and when to retrieve
- Dynamically optimizes retrieval for efficiency
- Enhances interpretability with transparent retrieval steps

Search-o1: Agentic Search-Enhanced Large Reasoning Models

Xiaoxi Li[1], Guanting Dong[1], Jiajie Jin[1], Yuyao Zhang[1], Yujia Zhou[2], Yutao Zhu[1], Peitian Zhang[1], Zhicheng Dou[1]*

[1]Gaoling School of Artificial Intelligence, Renmin University of China.
[2]Department of Computer Science and Technology, Tsinghua University.
*Corresponding Author.

EMNLP 2025

Presenter: Feng Qiao

Background

- **Adaptive-RAG** established that retrieval strategy should adapt to query complexity (simple vs complex).
- **Auto-RAG** empowered LLMs to autonomously control iterative retrieval (when and what to retrieve).
- **Search-o1**, which embeds agentic search into reasoning chains of LRMs.

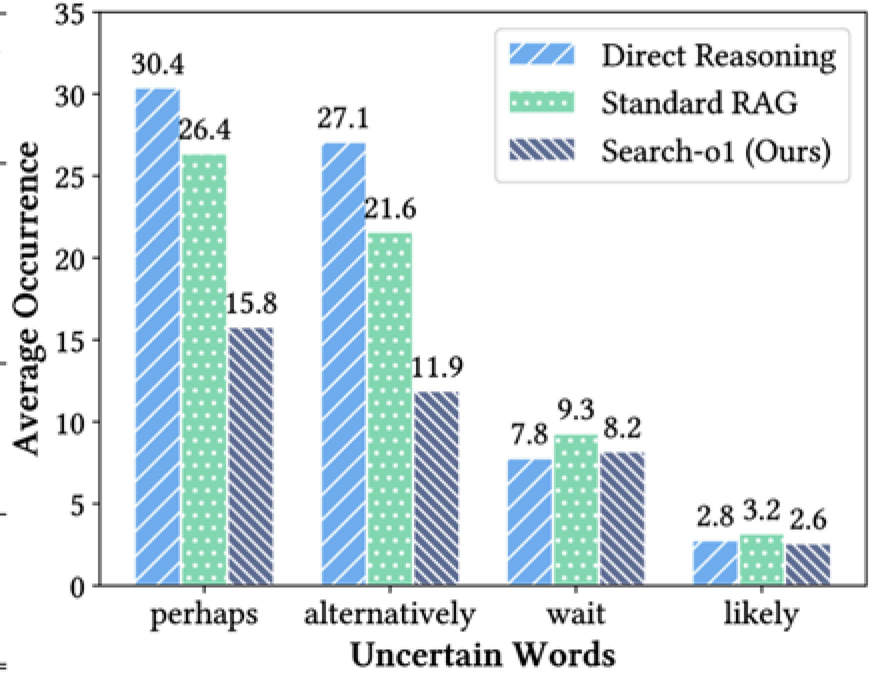
Background

Aspect	LLM (Large Language Model)	LRM (Large Reasoning Model)
Definition	Trained on massive text data to understand and generate natural language through next-token prediction.	Built on LLMs but optimized for logical reasoning, multi-step problem solving, and tool/search integration.
Advantages / Disadvantages	✅ Fluent, fast, creative, broad knowledge. ❌ May hallucinate, struggles with multi-step reasoning.	✅ Strong reasoning, math, and coding ability; can verify answers. ❌ Slower, more expensive, sometimes overthinks.
Examples	GPT-4, Claude 3, Gemini 1.5, LLaMA 3	o1 (OpenAI Reasoning), Claude 3.5 Sonnet (Pro Reasoning mode), Gemini 1.5 Pro (Reasoning Mode), DeepSeek-R1, Qwen-QwQ

Motivation

Large reasoning models (LRMs) can handle long reasoning chains in science/math/coding. But they suffer from knowledge insufficiency: when hitting uncertain knowledge points, the chain breaks or becomes unreliable

Cases of Model-Expressed Uncertainty
Wait, **perhaps** it's referring to dimethyl sulfone, but that doesn't seem right.
Alternatively, **perhaps** there's a mistake in my understanding of epistasis. Let me look up epistasis quickly. Epistasis is …
Alternatively, HBr could also abstract a hydrogen atom from the alkene, leading to a …
As I recall, Quinuclidine is a seven-membered ring with a nitrogen atom, **likely** not having the required symmetry.



Left: Examples of uncertain words identified during the reasoning process.
Right: Average occurrence of high-frequency uncertain words per output in the GPQA diamond set. Tested on QwQ-32B-Preview.

Challenges

But integrating retrieved knowledge into the LRM's reasoning process is not easy:

(1) Redundant Information in Retrieved Documents.

Retrieved documents are often lengthy and contain redundant information, directly inputting them into LRMs may disrupt the original coherence of reasoning and even introduce noise.

(2) **Limited Ability to Understand Long Documents.**

Most LRMs have been specifically aligned for complex reasoning tasks during the pre-training and fine-tuning stages. This focus has resulted in a degree of catastrophic forgetting in their general capabilities, ultimately limiting their long-context understanding of retrieved documents.

Problem Formulation

$$P(\mathcal{R}, a \mid I, q, \mathcal{D}) = \underbrace{\prod_{t=1}^{T_r} P(\mathcal{R}_t \mid \mathcal{R}_{<t}, I, q, \mathcal{D}_{<t})}_{\text{Reasoning Process}} \cdot \underbrace{\prod_{t=1}^{T_a} P(a_t \mid a_{<t}, \mathcal{R}, I, q)}_{\text{Answer Generation}},$$

$\mathcal{R}$ **Reasoning chain** — the full sequence of reasoning tokens representing the model's thought process.

a **Final answer**

I **Task instruction** — describes what kind of reasoning task or output format to follow.

q **Question**

$\mathcal{D}$ **Retrieved documents**

T_r **Length of reasoning chain**

T_a **Length of answer**

$\mathcal{R}_t$ **Current reasoning token**

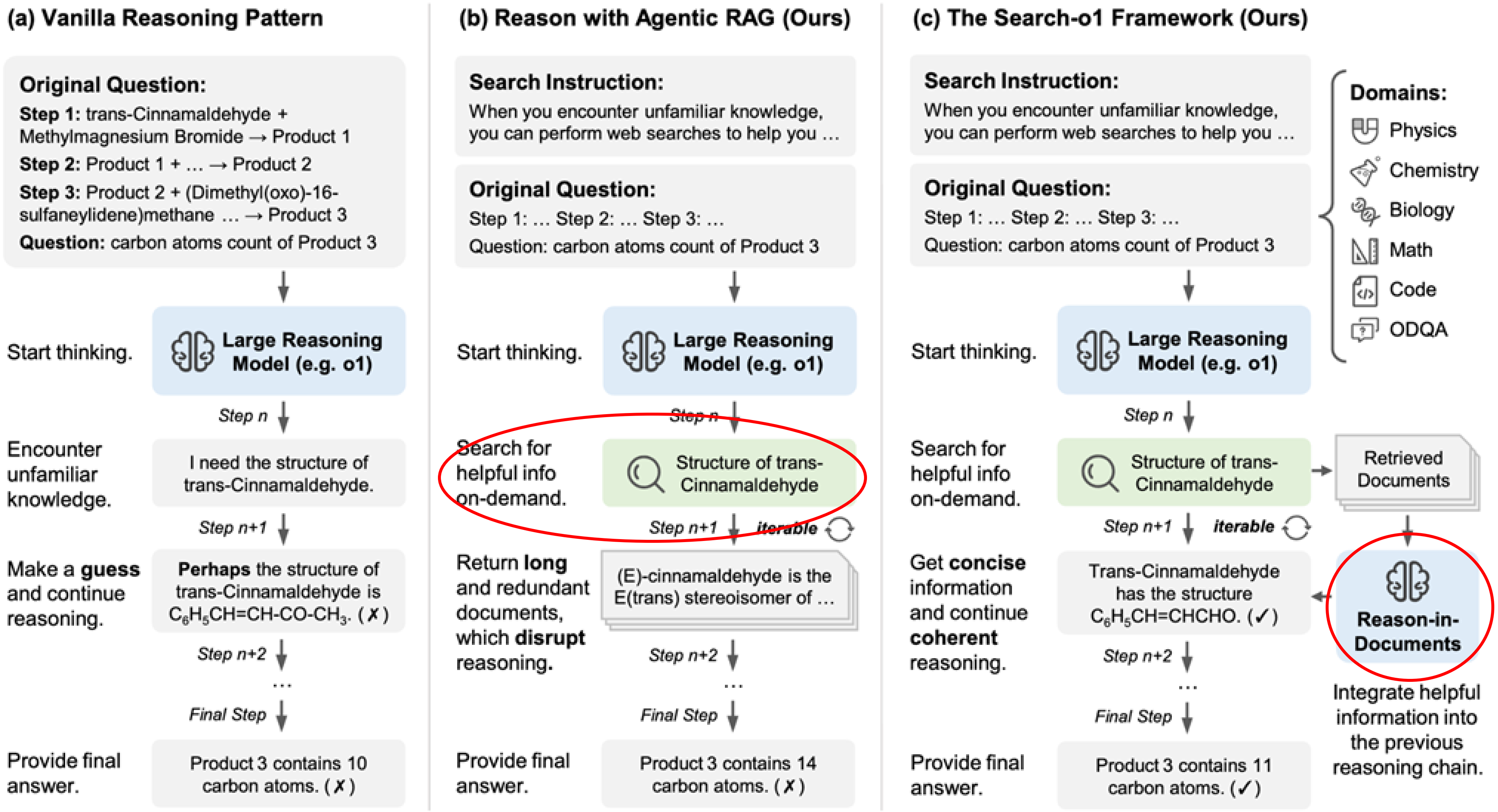
$\mathcal{R}_{<t}$ **Previous reasoning tokens**

$\mathcal{D}_{<t}$ **Retrieved docs up to current step**

a_t **Current answer token**

$a_{<t}$ **Previous answer tokens**

Overview of the Search-o1 Framework



Agentic RAG Mechanism

1. **Reasoning Stage**

The LRM generates reasoning tokens. When it encounters uncertainty, it decides to generate a search query.

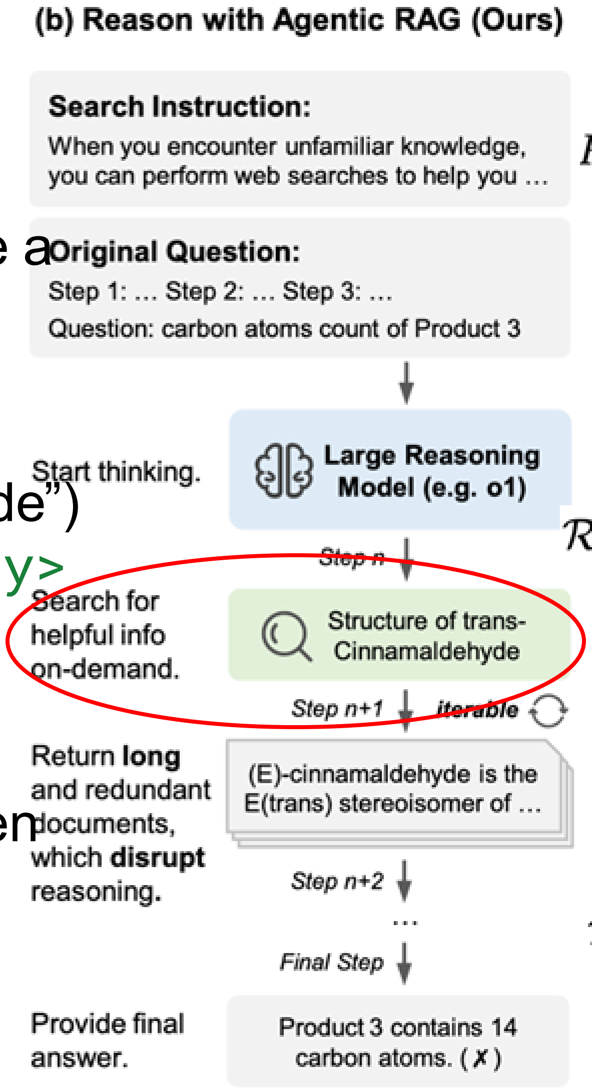
1. **Query Generation**

The model creates a query (e.g., “structure of trans-Cinnamaldehyde”) between <begin_search_query> and <end_search_query>.

1. **Retrieval**

The search system retrieves documents, then inserted between <begin_search_result> and <end_search_result>.

1. **Reasoning Resumes**



Formula 1: Search Query Generation

$$P(q_{\text{search}}^{(i)} \mid I, q, \mathcal{R}^{(i-1)}) = \prod_{t=1}^{T_q^{(i)}} P\left(q_{\text{search},t}^{(i)} \mid q_{\text{search},<t}^{(i)}, I, q, \mathcal{R}^{(i-1)}\right)$$

$T_q^{(i)}$ is the length of the i-th search query,

$q_{\text{search},t}^{(i)}$ denotes the token generated at step t of the i-th search query

$\mathcal{R}^{(i-1)}$ represents all the reasoning steps prior to the i-th search step

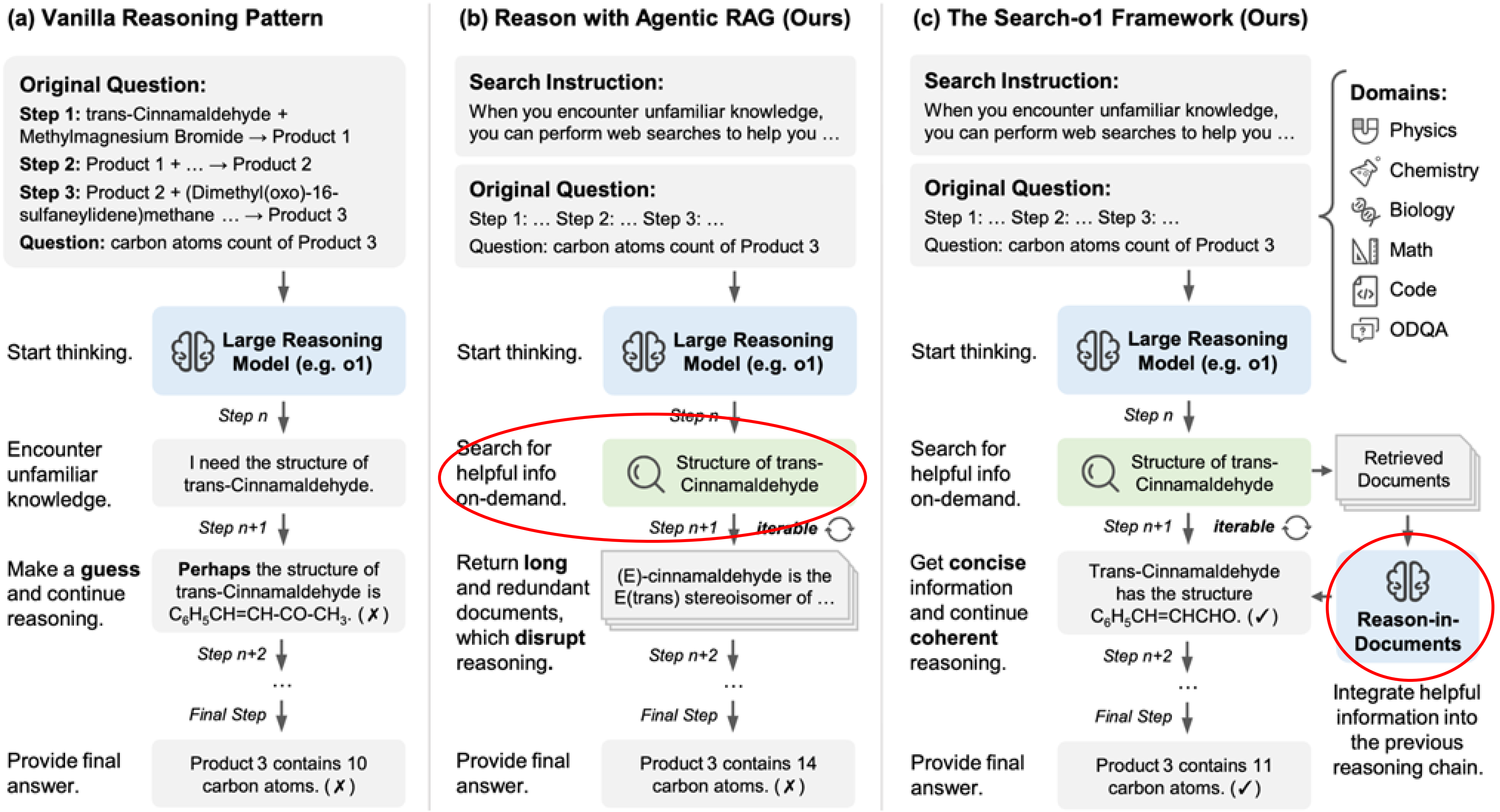
Formula 2: Knowledge Retrieval

$$\mathcal{D}^{(i)} = \texttt{Search}(q_{\text{search}}^{(i)})$$

Search() is the retrieval function

$\mathcal{D}^{(i)} = d_1^{(i)}, d_2^{(i)}, \ldots, d_{k_i}^{(i)}$ represents the set of top-k_i relevant documents

Overview of the Search-o1 Framework



Knowledge Refinement via Reason-in-Documents

Purpose: retrieved documents tend to be long and noisy; injecting raw docs risks disrupting reasoning or adding irrelevant context.

Steps:

1. Generates an **intermediate reasoning sequence** summarizing key info from retrieved docs

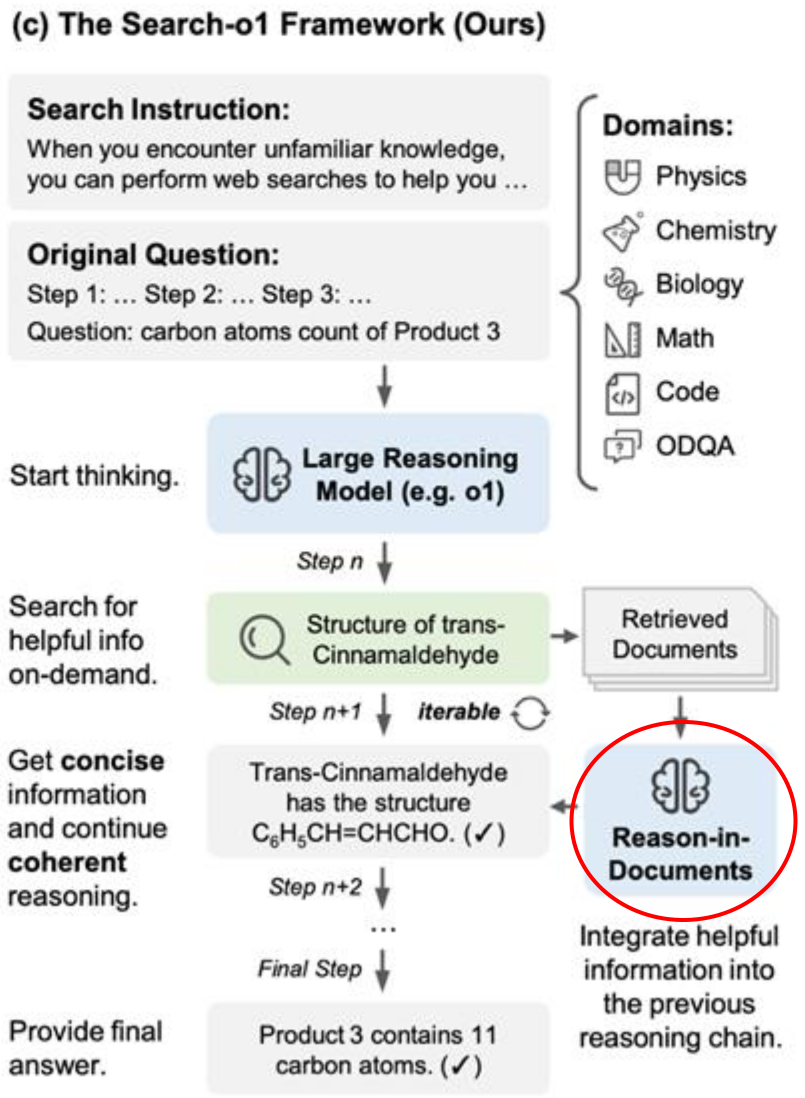
$$P(r_{\text{docs}}^{(i)} \mid \mathcal{R}^{(<i)}, q_{\text{search}}^{(i)}, \mathcal{D}^{(i)}) = \prod_{t=1}^{T_d^{(i)}} P\left(r_{\text{docs},t}^{(i)} \mid r_{\text{docs},<t}^{(i)}, \mathcal{R}^{(<i)}, q_{\text{search}}^{(i)}, \mathcal{D}^{(i)}\right)$$

1. Produces **refined knowledge** aligned with the model's current reasoning needs

$$P(r_{\text{final}}^{(i)} \mid r_{\text{docs}}^{(i)}, \mathcal{R}^{(<i)}, q_{\text{search}}^{(i)}) = \prod_{t=1}^{T_r^{(i)}} P\left(r_{\text{final},t}^{(i)} \mid r_{\text{final},<t}^{(i)}, r_{\text{docs}}^{(i)}, \mathcal{R}^{(<i)}, q_{\text{search}}^{(i)}\right)$$

1. Inserts the refined knowledge back into reasoning for

$$P(\mathcal{R}, a \mid I, q) = \prod_{t=1}^{T_r} P\left(\mathcal{R}_t \mid \mathcal{R}_{<t}, I, q, \{r_{\text{final}}^{(j)}\}_{j \le i(t)}\right) \cdot \prod_{t=1}^{T_a} P\left(a_t \mid a_{<t}, \mathcal{R}, I, q\right)$$



Experimental Setup

Aspect	Setting
Backbone model	**QwQ-32B-Preview**
Retriever & corpus	**Bing Web Search API** (region US-EN), **top-k=10**
Datasets (reasoning)	**GPQA** (Diamond & Extended), **MATH500**, **AMC2023**, **AIME2024**, **LiveCodeBench (Aug–Nov 2024, 112 problems)**
Datasets (open-domain QA)	**NQ, TriviaQA, HotpotQA, 2WIKI, MuSiQue, Bamboogle**
Metrics	**Pass@1/Accuracy**: how often the model's first generated answer is correct; **EM:** Exact Match. **F1:** partial overlap between the model's answer and the ground truth.

Experimental Setup - Evaluation datasets

Challenging Reasoning Tasks	**GPQA**	PhD-level science multiple-choice QA
	MATH500	**Math** problem
	AMC2023	Middle school **math**
	AIME2024	Advanced **math**
	LiveCodeBench	Programming and **code**
Open-Domain QA	**NQ (Natural Questions)**	Single-hop QA on real Google queries
	TriviaQA	Single-hop QA from trivia sources
	HotpotQA	Multi-hop QA across paragraphs
	2WikiMultiHopQA (2WIKI)	Multi-hop QA across two Wikipedia pages
	MuSiQue	Multi-hop QA with 2–4 hops
	Bamboogle	Multi-hop QA on hard real-world queries

Key Results

Table 1: Main results on challenging reasoning tasks, including PhD-level science QA, math, and code benchmarks. We report Pass@1 metric for all tasks. For models with 32B parameters, the best results are in **bold** and the second-best are underlined. Results from larger or non-proprietary models are in gray color for reference. Symbol "†" indicates results from their official releases.

Method	GPQA (PhD-Level Science QA)				Math Benchmarks			LiveCodeBench			
	Physics	Chemistry	Biology	Overall	MATH500	AMC23	AIME24	Easy	Medium	Hard	Overall
Direct Reasoning (w/o Retrieval)											
Qwen2.5-32B	57.0	33.3	52.6	45.5	75.8	57.5	23.3	42.3	18.9	14.3	22.3
Qwen2.5-Coder-32B	37.2	25.8	57.9	33.8	71.2	67.5	20.0	61.5	16.2	12.2	25.0
QwQ-32B	75.6	39.8	68.4	58.1	83.2	82.5	53.3	61.5	29.7	**20.4**	**33.0**
Qwen2.5-72B	57.0	37.6	68.4	49.0	79.4	67.5	20.0	53.8	29.7	24.5	33.0
Llama3.3-70B	54.7	31.2	52.6	43.4	70.8	47.5	36.7	57.7	32.4	24.5	34.8
DeepSeek-R1-Lite†	-	-	-	58.5	91.6	-	52.5	-	-	-	51.6
GPT-4o†	59.5	40.2	61.6	50.6	60.3	-	9.3	-	-	-	33.4
o1-preview†	89.4	59.9	65.9	73.3	85.5	-	44.6	-	-	-	53.6
Retrieval-augmented Reasoning											
RAG-Qwen2.5-32B	57.0	37.6	52.6	47.5	82.6	72.5	30.0	61.5	24.3	8.2	25.9
RAG-QwQ-32B	76.7	38.7	73.7	58.6	84.8	82.5	50.0	57.7	16.2	12.2	24.1
RAgent-Qwen2.5-32B	58.1	33.3	63.2	47.0	74.8	65.0	20.0	57.7	24.3	6.1	24.1
RAgent-QwQ-32B	76.7	46.2	68.4	61.6	85.0	**85.0**	**56.7**	**65.4**	18.9	12.2	26.8
Retrieval-augmented Reasoning with Reason-in-Documents											
Search-o1 (Ours)	**77.9**	**47.3**	**78.9**	**63.6**	**86.4**	**85.0**	**56.7**	57.7	**32.4**	**20.4**	**33.0**

Key Results-open-diamin datasets

Table 3: Performance comparison on open-domain QA tasks, including single-hop QA and multi-hop QA datasets. For models with 32B parameters, the best results are in **bold** and the second-best are underlined. Results from larger models are in gray color for reference.

Method	Single-hop QA				Multi-hop QA							
	NQ		TriviaQA		HotpotQA		2WIKI		MuSiQue		Bamboogle	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Direct Reasoning (w/o Retrieval)												
Qwen2.5-32B	22.8	33.9	52.0	60.3	25.4	34.7	29.8	36.3	8.4	18.0	49.6	63.2
QwQ-32B	23.0	33.1	53.8	60.7	25.4	33.3	34.4	40.9	9.0	18.9	38.4	53.7
Qwen2.5-72B	27.6	41.2	56.8	65.8	29.2	38.8	34.4	42.7	11.4	20.4	47.2	61.7
Llama3.3-70B	36.0	48.7	68.8	76.8	37.8	49.1	46.0	54.2	14.8	23.6	54.4	67.8
Retrieval-augmented Reasoning												
RAG-Qwen2.5-32B	33.4	<u>49.3</u>	**65.8**	**79.2**	38.6	50.4	31.6	40.6	10.4	19.8	52.0	66.0
RAG-QwQ-32B	29.6	44.4	<u>65.6</u>	<u>77.6</u>	34.2	46.4	35.6	46.2	10.6	20.2	<u>55.2</u>	<u>67.4</u>
RAgent-Qwen2.5-32B	32.4	47.8	63.0	72.6	<u>44.6</u>	<u>56.8</u>	55.4	69.7	13.0	25.4	54.4	66.4
RAgent-QwQ-32B	<u>33.6</u>	48.4	62.0	74.0	43.0	55.2	**58.4**	<u>71.2</u>	<u>13.6</u>	<u>25.5</u>	52.0	64.7
Retrieval-augmented Reasoning with Reason-in-Documents												
Search-o1 (Ours)	**34.0**	**49.7**	63.4	74.1	**45.2**	**57.3**	<u>58.0</u>	**71.4**	**16.6**	**28.2**	**56.0**	**67.8**

Other results

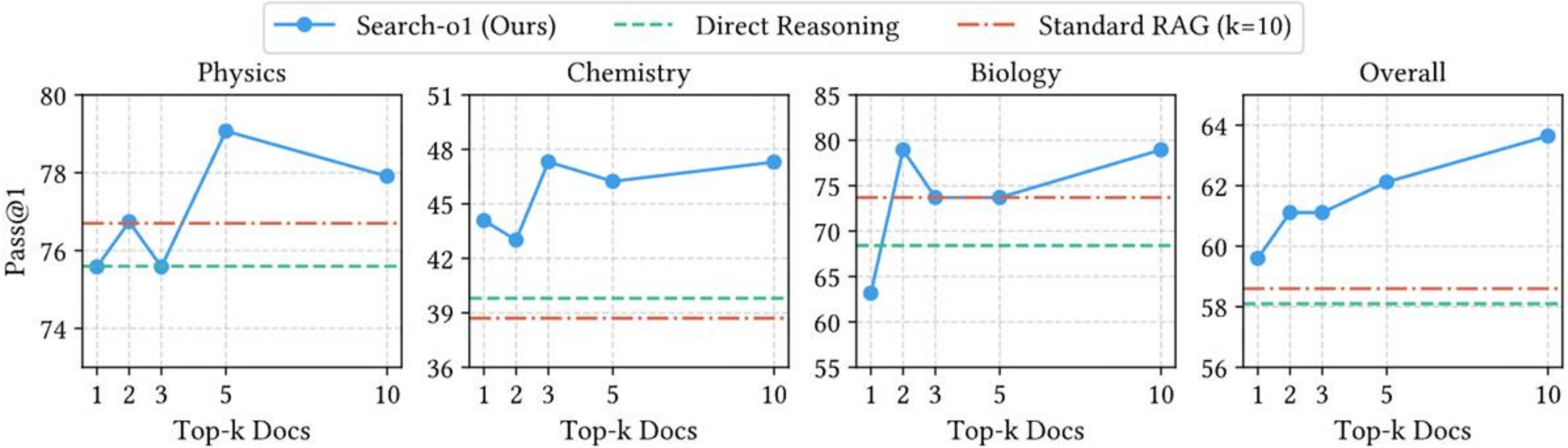


Figure 3: Scaling analysis of top-k retrieved documents utilized in reasoning. All results are based on QwQ-32B-Preview model.

Discussion & Implications

- Implications for future large reasoning models:
  a. Instead of “just bigger models”, integrate retrieval/search as part of the reasoning loop.
  b. LMs becoming more agentic: deciding when to ask for help (search) rather than trying to answer from internal memory only.
  c. Could reduce hallucinations by providing grounded retrieval when needed.
- Limitations: retrieval latency, dependency on quality of document corpora, potential for retrieving misleading docs.
- Open questions: how to choose retrieval trigger thresholds; how to handle contradictory retrieved info; cost of real-time search at inference scale.

Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning

Bowen Jin[1], **Hansi Zeng**[2], **Zhenrui Yue**[1], **Jinsung Yoon**[3], **Sercan Ö. Arık**[3], **Dong Wang**[1], **Hamed Zamani**[2], **Jiawei Han**[1]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign
[2] Center for Intelligent Information Retrieval, University of Massachusetts Amherst
[3] Google Cloud AI Research

COLM 2025

co-presenters: Alexis Liao, Xuanzhen Lao, Feng Qiao

The Core Problem & The Goal

- **The Problem:** Prompting an LLM to use a search engine (like in ReAct or RAG) is a good start, but it's *suboptimal*. The LLM doesn't *learn* how to get better at searching or reasoning from its mistakes.
- **The Goal:** Move from *prompting* to *training*
- **Our Method (Search-R1):** Use **Reinforcement Learning (RL)** to train the LLM.
  - The LLM learns *when* to search, *what* to search for, and *how* to use the results to reason, all by trial and error
  - It learns to maximize a "reward" signal

The Framework: RL + Search Engine

We formulate this as an RL problem. The main components are:

- **Policy LLM:** This is the "agent" or the "brain" we are training. It generates the text, including its thoughts and search queries
- **Search Engine:** This is treated as part of the "environment." The LLM can't change it, but it can interact with it
- **Reward:** This is the simple "score" the LLM gets at the end. Did it get the right answer?

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(\cdot|x;\mathcal{R})} \left[r_\phi(x,y)\right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x;\mathcal{R}) \,||\, \pi_{\mathrm{ref}}(y \mid x;\mathcal{R})\right],$$

The Interaction Loop (The "Rollout")

How does the LLM actually interact with the search engine? It uses special tokens to structure its thinking and actions.

1. `<think>`...`</think>`: The LLM first reasons about the problem
2. `<search>`...`</search>`: If it needs information, it generates a query inside these tags
3. **System Pauses:** The system detects the `</search>` tag, pauses the LLM, and runs the query
4. `<information>`...`</information>`: The search results are inserted back into the context, wrapped in these tags
5. **Loop:** The LLM sees the new info and goes back to step 1 (`<think>...`)
6. `<answer>`...`</answer>`: When it's confident, it provides the final answer

This is all part of one continuous generation sequence.

The Training Template (How We Start)

To get the LLM started, we use a very simple template. It only enforces the *structure*, not the *content* of the reasoning.

Answer the given question. You must conduct reasoning inside <think> and </think> first every time you get new information. After reasoning, if you find you lack some knowledge, you can call a search engine by <search> query </search>, and it will return the top searched results between <information> and </information>. You can search as many times as you want. If you find no further external knowledge needed, you can directly provide the answer inside <answer> and </answer> without detailed illustrations. For example, <answer> xxx </answer>. Question: question.

Table 1: Template for SEARCH-R1. question will be replaced with the specific question during training and inference.

This lets the RL process discover the *best* reasoning and search strategies on its own, rather than us telling it *how* to think.

Key Innovation: Retrieved Token Masking

This is the most important technical detail for making the training stable.

- **Problem:** The text the LLM sees is a mix of its *own* generated tokens (thoughts, queries) and tokens *from the search engine* (retrieved information)
- **Why is this bad?** We only want to train the LLM on its *own decisions*. We shouldn't reward or penalize it for the content of the search results, which it didn't write
- **Solution: Loss Masking:**
  - During the RL update, we set the loss to **zero** for all tokens inside `<information>...</information>`
  - The model is *only* trained on the tokens it generates: its thoughts, its search queries, and its final answer
  - This prevents the model from being trained on "noise" and makes the training process much more stable

The Reward Model: Simple is Better

How do we "score" the LLM's final answer? The paper makes a very important and simple choice.

- **No complex, step-by-step reward.** We don't try to judge if each thought or search query was "good."
- **No trained reward model.** We don't use another LLM to grade the answer.
- **Just a Simple, Rule-Based, Outcome Reward:**
    - `Reward = 1` if the text inside <answer>...</answer> is an **Exact Match (EM)** with the ground truth.
    - `Reward = 0` if it is not.

This is powerful. It means the model can learn a complex, multi-step reasoning and search behavior from a simple "You got it right" or "You got it wrong" signal.

$$r_\phi(x, y) = \mathrm{EM}(a_{\text{pred}}, a_{\text{gold}}),$$

Summary of Method & Hand-off

So, to summarize the Search-R1 method:

1. **Framework:** We use Reinforcement Learning (PPO or GRPO) to train an LLM to treat a search engine as part of its environment
2. **Interaction:** The LLM learns to generate special tokens—<think>, <search>, and <answer>—to build a reasoning and action loop
3. **Key Technique:** We use **Retrieved Token Masking** to ensure the LLM only learns from its own generated tokens, which stabilizes training
4. **Feedback:** The entire process is guided by a simple, **outcome-based Exact Match reward**, proving that a complex behavior can be learned from a simple signal

Experimental setups

Training Data	NQ + HotpotQA (merge their training sets)
Models Used	Qwen-2.5-3B (Base/Instruct), Qwen-2.5-7B (Base/Instruct) (Yang et al., 2024)
Retriever & Corpus	2018 Wikipedia dump (Karpukhin et al., 2020) E5 (Wang et al., 2022)
RL Settings	PPO (default)
Evaluation metric	Exact Match (EM)

Experimental setups

Evaluation Dataset: **<u>In domain</u>**

Type	Dataset	Features
General QA	**NQ** (Natural Questions)	Answer real Google search queries using Wikipedia
	TriviaQA	Answer factoid trivia questions
	PopQA	Evaluate factual and pop-culture knowledge
Multi-Hop QA	**HotpotQA**	Multi-hop reasoning with supporting facts
	2WikiMultiHopQA	Multi-hop reasoning across two Wikipedia pages
	MuSiQue	Decompose complex questions into sub-questions
	Bamboogle	Real-time web search required

Results-main results- Qwen2.5-7b-Base

Table 2: Main results. The best performance is set in bold. †/⋆ represents in-domain/out-domain datasets.

Methods	General QA			Multi-Hop QA				
	NQ†	TriviaQA⋆	PopQA⋆	HotpotQA†	2wiki⋆	Musique⋆	Bamboogle⋆	Avg.
Qwen2.5-7b-Base/Instruct								
Direct Inference	0.134	0.408	0.140	0.183	0.250	0.031	0.120	0.181
CoT	0.048	0.185	0.054	0.092	0.111	0.022	0.232	0.106
IRCoT	0.224	0.478	0.301	0.133	0.149	0.072	0.224	0.239
Search-o1	0.151	0.443	0.131	0.187	0.176	0.058	0.296	0.206
RAG	0.349	0.585	0.392	0.299	0.235	0.058	0.208	0.304
SFT	0.318	0.354	0.121	0.217	0.259	0.066	0.112	0.207
R1-base	0.297	0.539	0.202	0.242	0.273	0.083	0.296	0.276
R1-instruct	0.270	0.537	0.199	0.237	0.292	0.072	0.293	0.271
Rejection Sampling	0.360	0.592	0.380	0.331	0.296	0.123	0.355	0.348
Search-R1-base	**0.480**	**0.638**	**0.457**	**0.433**	0.382	**0.196**	**0.432**	**0.431**
Search-R1-instruct	0.393	0.610	0.397	0.370	**0.414**	0.146	0.368	0.385

Results-main results-Qwen2.5-3b-Base

Table 2: Main results. The best performance is set in bold. †/⋆ represents in-domain/out-domain datasets.

Methods	General QA			Multi-Hop QA				
	NQ†	TriviaQA⋆	PopQA⋆	HotpotQA†	2wiki⋆	Musique⋆	Bamboogle⋆	Avg.
Qwen2.5-3b-Base/Instruct								
Direct Inference	0.106	0.288	0.108	0.149	0.244	0.020	0.024	0.134
CoT	0.023	0.032	0.005	0.021	0.021	0.002	0.000	0.015
IRCoT	0.111	0.312	0.200	0.164	0.171	0.067	0.240	0.181
Search-o1	0.238	0.472	0.262	0.221	0.218	0.054	**0.320**	0.255
RAG	0.348	0.544	0.387	0.255	0.226	0.047	0.080	0.270
SFT	0.249	0.292	0.104	0.186	0.248	0.044	0.112	0.176
R1-base	0.226	0.455	0.173	0.201	0.268	0.055	0.224	0.229
R1-instruct	0.210	0.449	0.171	0.208	0.275	0.060	0.192	0.224
Rejection Sampling	0.294	0.488	0.332	0.240	0.233	0.059	0.210	0.265
Search-R1-base	**0.406**	**0.587**	**0.435**	0.284	0.273	0.049	0.088	0.303
Search-R1-instruct	0.341	0.545	0.378	**0.324**	**0.319**	**0.103**	0.264	**0.325**

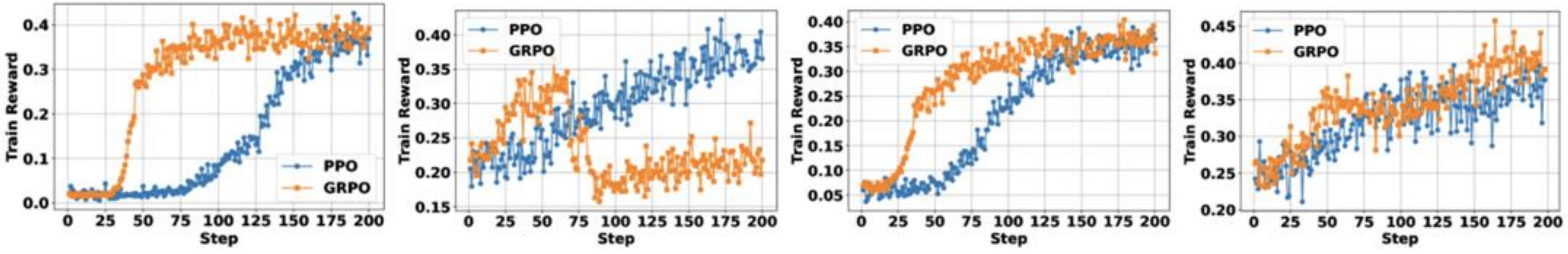
Results-compare different RLs

Table 3: The performance results of SEARCH-R1 with PPO and GRPO on seven datasets.

Method	NQ	TriviaQA	PopQA	HotpotQA	2wiki	Musique	Bamboogle	Avg.
Qwen2.5-7b-Base/Instruct								
SEARCH-R1-base (GRPO)	0.395	0.560	0.388	0.326	0.297	0.125	0.360	0.350
SEARCH-R1-instruct (GRPO)	0.429	0.623	0.427	0.386	0.346	0.162	0.400	0.396
SEARCH-R1-base (PPO)	**0.480**	**0.638**	**0.457**	**0.433**	0.382	**0.196**	**0.432**	**0.431**
SEARCH-R1-instruct (PPO)	0.393	0.610	0.397	0.370	**0.414**	0.146	0.368	0.385
Qwen2.5-3b-Base/Instruct								
SEARCH-R1-base (GRPO)	**0.421**	0.583	0.413	0.297	0.274	0.066	0.128	0.312
SEARCH-R1-instruct (GRPO)	0.397	0.565	0.391	**0.331**	0.310	**0.124**	0.232	**0.336**
SEARCH-R1-base (PPO)	0.406	**0.587**	**0.435**	0.284	0.273	0.049	0.088	0.303
SEARCH-R1-instruct (PPO)	0.341	0.545	0.378	0.324	**0.319**	0.103	**0.264**	0.325

Analysis-Different RL methods: PPO vs. GRPO

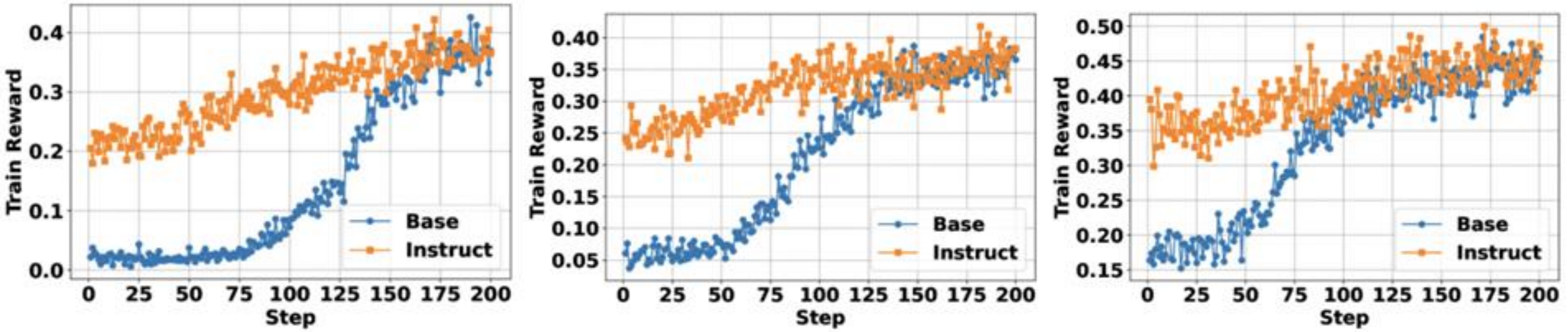
- GRPO converges faster than PPO across all cases.
- PPO is more stable
- Both reach similar final rewards
- Performance comparison: GRPO slightly outperforms PPO on most datasets (+3–5 points avg).



(a) LLaMA3.2-3b-base (b) LLaMA3.2-3b-it (c) Qwen2.5-3b-base (d) Qwen2.5-3b-it

Base vs Instruct Models

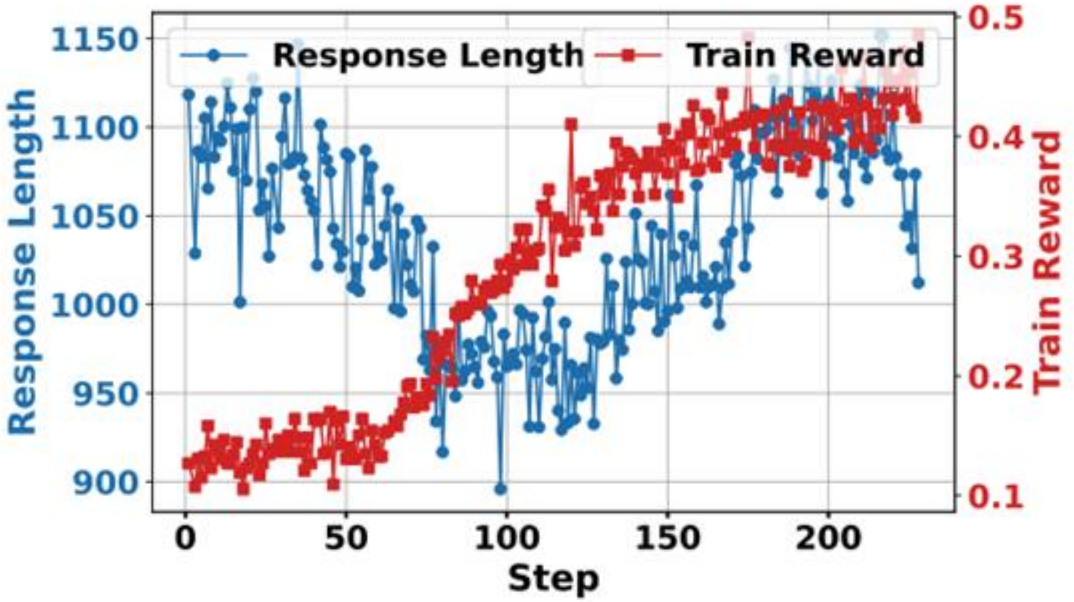
- Instruct models start better and learn faster
- Base models catch up after RL training — reinforcement learning bridges the gap.
- Final performances are similar, showing Search-R1 is effective for both types.



(a) LLaMA3.2-3b-base/instruct (b) Qwen2.5-3b-base/instruct (c) Qwen2.5-7b-base/instruct

Response Length Study

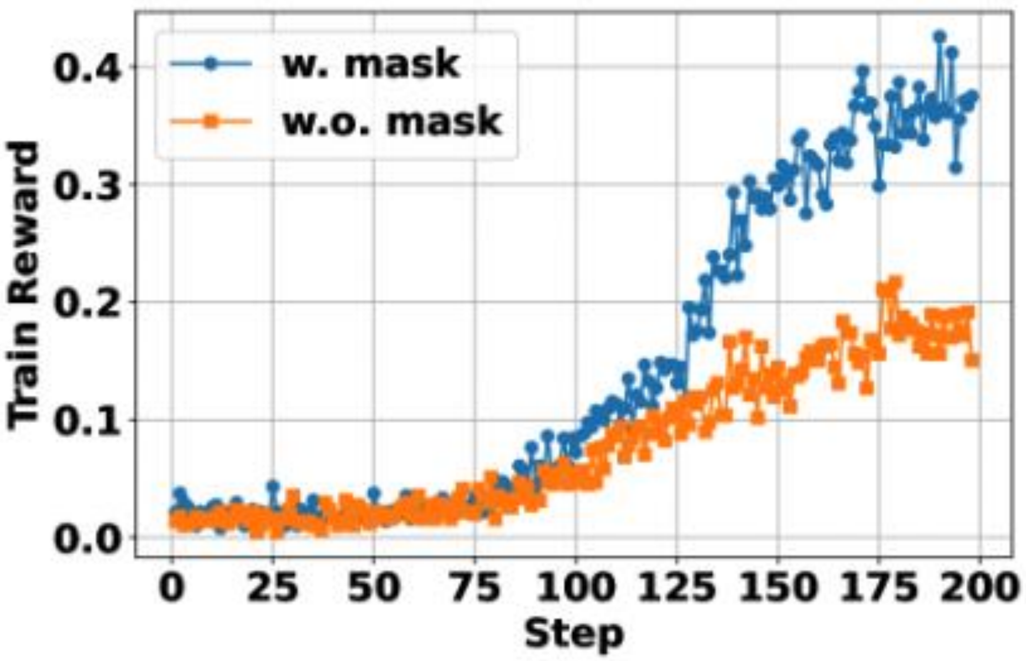
- **Early stage (0-100 steps):** Responses as model learns to use search engine.
- **Late stage (>130 steps):** Length stabilizes ≈ 500 tokens → converged behavior.
- Training reward increases steadily, matching the response length curve.



(c) Response length

Loss Masking Effect

- **Without masking**, the model overfits to retrieved text and becomes unstable.
- **With masking**, it focuses on LLM-generated tokens, leading to stable and better results.
- Average performance improves from 0.147 to 0.305.



(b) Loss mask dynamics study

Conclusion

- Search-R1enables LLMs to combine reasoning with real-time search, overcoming the limitations of RAG and Tool-Use methods.

- **Proven Results:** Search-R1 enhanced the LLM's ability to solve complex reasoning tasks that require external knowledge.