AirQA: A Comprehensive QA Dataset for AI Research with Instance-Level Evaluation

Tiancheng Huang[1] **Ruisheng Cao**[1] **Yuxin Zhang**[1] **Zhangyi Kang**[1] **Zijian Wang**[1]

Chenrun Wang[1] **Yijie Luo**[1] **Hang Zheng**[1] **Lirong Qian**[1] **Lu Chen**[12†] **Kai Yu**[12†]

[1]MoE Key Lab of Artificial Intelligence, Shanghai, China
X-LANCE Lab, School of Computer Science, Shanghai Jiao Tong University, Shanghai, China
Jiangsu Key Lab of Language Computing, Suzhou, China
[2]Suzhou Laboratory, Suzhou, China
{htc981,chenlusz,kai.yu}@sjtu.edu.cn

Abstract

The growing volume of academic papers has made it increasingly difficult for researchers to efficiently extract key information. While large language models (LLMs) based agents are capable of automating question answering (QA) workflows for scientific papers, there still lacks a comprehensive and realistic benchmark to evaluate their capabilities. Moreover, training an interactive agent for this specific task is hindered by the shortage of high-quality interaction trajectories. In this work, we propose AirQA, a human-annotated comprehensive paper QA dataset in the field of artificial intelligence (AI), with 13,948 papers and 1,246 questions, that encompasses multi-task, multi-modal and instance-level evaluation. Furthermore, we propose ExTrActor, an automated framework for instruction data synthesis. With three LLM-based agents, ExTrActor can perform example generation and trajectory collection without human intervention. Evaluations of multiple open-source and proprietary models show that most models underperform on AirQA, demonstrating the quality of our dataset. Extensive experiments confirm that ExTrActor consistently improves the multi-turn tool-use capability of small models, enabling them to achieve performance comparable to larger ones.

1 Introduction

With the explosion of artificial intelligence (AI) publications, researchers must spend a significant amount of time reading lengthy papers just to locate a highly specific piece of information, which is both tedious and inefficient. The advent of large language models (LLMs), especially their remarkable reasoning and planning capabilities [1, 11, 15, 38], has made it possible to automate the workflow of precise retrieval and question answering (QA) for academic papers [13, 27, 33]. Despite recent advances, there remains a notable absence of a comprehensive and realistic benchmark, which covers diverse question types and multi-modal abilities. And training an interactive QA agent that focuses on such task is difficult due to the scarcity of high-quality domain-specific trajectories.

Previous QA datasets on scientific papers usually focus on one narrow question type, such as querying technical details about a single paper [9, 21, 28, 32], questions spanning across multiple

[*]Our code and data will be made publicly available soon.
[†]The corresponding authors are Lu Chen and Kai Yu.

Preprint. Under review.

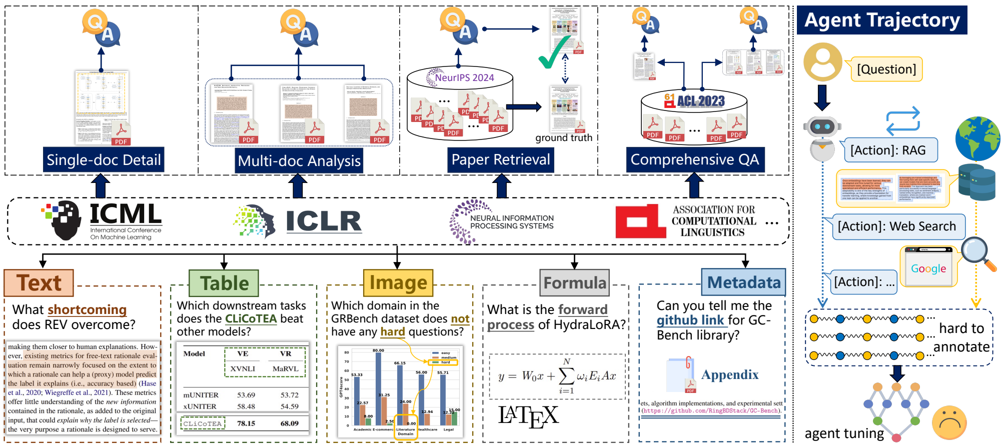

Figure 1: Left: An overview of the four question types and five element categories in our AirQA dataset. Right: An illustration of the bottleneck in multi-turn tool-use trajectory collection.

documents following a rule-constructed two-hop pattern [22], or aiming at the common paper retrieval requirements [2, 13]. Accordingly, the evaluation function is usually tailored for one restricted type and lacks generalization to others. For example, M3SciQA [22] designed one LLM-based prompt for long-form string evaluation with the reference answer, which is highly empirical and only serves its specific question type. On the other hand, most benchmark owners overly pre-process raw papers, and merely provide the cleaned text format for uniform input. This common practice deviates from realistic scenarios, where real-world users may query other hyper-textual elements (illustrated in the bottom part of Figure 1) embedded in the raw PDF documents, such as figures, tables, formula, metadata, or even different combinations of them.

While tackling QA on academic papers, trivial methods (e.g. provide titles and abstracts alongside the question [9, 32]) will easily fail due to context limitation, as the scaling of papers augment from a single paper to the entire conference volume. More advanced approaches adopt the popular RAG framework [6, 12, 17], but are not applicable in questions that require multi-turn reasoning over various chunked snippets. Meanwhile, interactive QA agents, which can predict executable retrievals or function-calling actions and interact with the outer environment for external knowledge, exhibit significant potential in handling long-context multi-hop scenarios, making it a good choice for scientific QA under realistic, complicated and universal settings [13, 26, 31]. Unfortunately, manually annotating task-specific trajectories of interactions with the environment is both time-consuming and expensive, requiring domain expertise, while simple data generation with LLMs can't faithfully synthesize $(action, observation)$ sequences with internal coherence and dependencies. As a result, the paucity of high-quality trajectory prevents the post-training of an effective QA agent.

To resolve the aforementioned bottlenecks, we propose a human-annotated multi-modal multi-task **A**rtificial **I**ntelligence **R**esearch **Q**uestion **A**nswering dataset, AirQA, which encompasses $1{,}246$ examples and $13{,}948$ papers, aiming at evaluating an agent's research capabilities in realistic scenarios. As illustrated in Figure 1, our dataset contains 4 different question types and 5 different element categories, with 19 parameterized Python functions to support customized evaluation. Furthermore, to advocate agentic model post-training, we propose a multi-agent framework, ExTrActor, for instruction data synthesis, which includes an explorer that generates natural language question-answer pairs based on contexts from the papers, a tracker that rewrites QA pairs into properly formatted examples, and an actor that interacts with the environment to collect trajectories of the examples.

We evaluate a wide range of open-source and proprietary LLMs on different baselines. Performances show that, though given several external information sources, LLMs struggle on our AirQA dataset, with the best model scoring only 44.14% overall, indicating that existing workflows are still underdeveloped. With the proposed ExTrActor framework, we fine-tune models of different sizes from the Qwen2.5 family [29]. Results show that, with just $4{,}000$ interaction trajectories, fine-tuned 7B model achieves a performance comparable to untrained 14B model. Extensive experiments demonstrate that, the accuracy raises consistently as data scales up, highlighting the scalability of our framework.

To summarize, our contributions are threefold:

- We propose AirQA, a human-annotated multi-modal multi-task multi-paper QA dataset with function-based instance-specific evaluations. To the best of our knowledge, AirQA is the first dataset that encompasses multiple question types, also the first to bring function-based evaluation into QA domain, enabling convenient and systematic assessment of research capabilities.
- We introduce EXTRACTOR, a document-based framework aiming at the synthesis of QA examples, interaction trajectories and instruction data, serving as an empirical method for improving the agent's multi-turn tool-using ability without the involvement of manual annotation.
- We evaluate various LLMs and different QA baselines on our AirQA dataset, demonstrating the quality of our dataset, and indicating the insufficiency of current methods. Extensive experiments on instruction tuning reveal that, small models significantly benefit from our synthetic instruction data, validating the effectiveness of our proposed EXTRACTOR framework.

2 The AirQA Dataset

In this section, we introduce the task definition, the evaluation metrics, the construction and the statistics of our AirQA dataset.

2.1 Task Definition

To more effectively evaluate existing models and methods across a broader range of tasks, rather than limiting assessment to individual tasks, we carefully analyze real-world AI research scenarios, and systematically design the following four question types in our AirQA dataset to cover them up:

Table 1: Examples of different question types from our AirQA dataset.

Type	Question	Answer Format
single	Which downstream tasks does the CLiCoTEA outperform other models in terms of zero-shot performance on the IGLUE benchmark?	Your answer should be a Python list of strings, every string is the abbreviation of a downstream task type mentioned in the paper.
multiple	According to this survey, what're the three most recent decoder-only LLMs for NL2Code? How many programming languages do their training datasets each contain?	Your answer should be a Python dictionary of 3 key-value pairs, where each key is a string, the LLM, and each value is the number of programming languages.
retrieval	Which paper unifies reinforcement learning and imitation learning methods under a dual framework?	Your answer should be the exact title of the paper WITHOUT ANY OTHER EXPLANATION.
comprehensive	Among the text-to-SQL papers in ACL 2023, which one achieves the best testsuite accuracy on the SPIDER dataset? Tell me the paper title and corresponding test accuracy.	Your answer should be a Python list of length two, with the first one being the title string and the second one being a float, the accuracy rounded to 3 decimals.

Single-doc Detail Querying detailed information from a specific paper. Besides text, we also explore different textual and non-textual aspects including table, image, formula and metadata to cover all elements that may appear in a scientific paper. We showcase one example for each category in Figure 1. Notably, a question may belong to multiple categories, requiring diverse capabilities.

Multiple-doc Analysis Posing questions across multiple papers. A simple idea for constructing multiple-doc questions is to merely combine several single-doc questions, but this method overlooks the possible relations between different papers, which are actually what researchers pay more attention to. To imitate the real scenes where researchers scan across several documents to find the answer to a question, we propose two paradigms: 1) compare same aspects of different papers, and 2) find subtle points that are not fully illustrated in one paper, and explore the details in the papers it cites.

Paper Retrieval Retrieving papers from a specific conference in a particular year, based on the description. Considering the search scale, while some argues that retrieval on a fixed corpus is not suitable as performance proxies for real scientific research tasks [33], we insist that a dataset cannot contain an infinite number of papers. A large number of new papers are produced every day, and without limitation, the answer would be ambitious, making the evaluation unfair. Only retrieval on a fixed corpus can ensure the objectivity of the dataset. Among these questions, 240 are directly transformed from author-written questions in the LitSearch [2] dataset with rule-based conversion.

Comprehensive QA A combination of the three aforementioned question types. Specifically, a comprehensive QA question may combine a paper retrieval question with either a single-doc question or a multiple-doc question. As an integrated task, this combination is designed for scenarios in which the user cannot directly provide the paper or has forgotten the specific paper to which the question refers, but recalls certain key points, thereby enabling retrieval. The process of addressing such a question can be divided into two main stages: retrieving the paper based on its description, and answering the detailed question.

We also exhibit one example for each question type in Table 1. For brevity, in the following sections, we refer to the four question types as single, multiple, retrieval and comprehensive respectively. For retrieval and comprehensive questions, to ensure uniqueness of the retrieved paper, and to avoid ambiguity, we limit the scope of retrieval to be one of ACL2023, ICLR2024 and NeurIPS2024.

2.2 Evaluation Method

As mentioned in Section 5, most previous QA datasets depend on linguistics metrics and LLMs for assessment, favoring semantic coherence over factual correctness, which holds little value under current circumstances. While in our AirQA dataset, we mainly focus on judging the correctness of the answer as objective as possible. We notice that, though the answers to different questions vary, they share common features. For example, when answering questions related to quantitative comparison, we only care about the number itself, rather than wether LLMs form a complete sentence. In this case, the number is the "scoring point" of this question, which directly determines the quality of the answer. Inspired by the instruction following ability of LLMs, we adopt output reformatting by providing an answer format along with the question (as shown in Table 1), such as, "*Your answer should be a Python list of two floats, each rounded to 2 decimal places.*". In this way, we guide LLMs to output the scoring points we primarily concern with, benefiting the following evaluation.

To evaluate scoring points of different kinds, we design 19 Python functions. To support example-specific assessment, we complement them with optional keyword arguments (e.g. `ignore_order` for list comparison). For each evaluation function, the final result will be either 0 or 1, representing wrong and right. Based on whether they utilize LLMs for semantic judgment or not, and their functionalities, these functions can be classified into two types and six subtypes as shown in Table 8. An evaluation function is subjective if it involves LLMs, and is objective if not. Specifically, for logical functions, which combine multiple functions in one evaluation, the evaluation is classified as subjective as long as there is one subjective function. For subjective functions, we select GPT-4o-mini-2024-07-18 as the backbone model for its relative stability. More details of the functions can be found in App. A.3.

2.3 Dataset Construction

Annotators To ensure the professionalism of the dataset, we employ 26 students with expertise in artificial intelligence. Their task is threefold: 1) read a paper they are interested in, 2) pose an answerable question based on the textual and non-textual content of the paper (and additional papers if needed) they read, in accordance with the aforementioned question types, 3) wrap the question, the evaluation function, and other necessary information into an example file, as presented in App. A.1. The example file is then sent into an automated inspection pipeline, and annotators are asked to rewrite unqualified examples.

Paper Collection Due to the professional background of the annotators, all papers are selected from the field of artificial intelligence to ensure accurate comprehension of the content. Most papers utilized in our AirQA dataset can be downloaded from `arXiv` (see App. G for more details). To facilitate reproduction, we assign an uuid for each used paper based on its title and its conference. We also generate a metadata file for each paper, containing the title, the abstract, the URL where the paper is downloaded, and other information. For further illustration, please refer to App. A.2.

2.4 Dataset Statistics

Example Classification We classify the examples in the AirQA dataset into four question types, five element categories and two evaluation types as discussed before. Table 2 shows that, the example numbers of the four question types are relatively balanced, while approximately half of the examples involve at least one element other than text (we classify an example as text if and only if it doesn't include any other elements, and an example can belong to more than one categories). Regarding evaluation functions, from Figure 2, we can observe that most evaluations utilized are objective, indicating that they don't require LLMs, which demonstrates the cost-effectiveness of our dataset.

Paper Usage To ensure the objectiveness of retrieval and comprehensive questions, we limit the retrieval scale to be one of ACL2023, ICLR2024 and NeurIPS2024. Besides including all papers from these three conferences in our collection, we also utilize another 707 papers in the examples, summing up a total of 13,948 papers. As shown in Table 2, in average, an example involve 1.63 papers, indicating the diversity of our dataset.

Table 2: Statistics of examples. For the last two statistics, we only consider single and multiple questions.

Statistics	Number
Question Type	
- single	351(28%)
- multiple	323(26%)
- retrieval	288(23%)
- comprehensive	284(23%)
Element Category	
- text	621(50%)
- table	213(17%)
- image	207(17%)
- formula	127(10%)
- metadata	122(10%)
Overall	**1246(100%)**
Avg. question length	34.84
Max. question length	118
Avg. # papers per example	1.63
Max. # papers per example	7

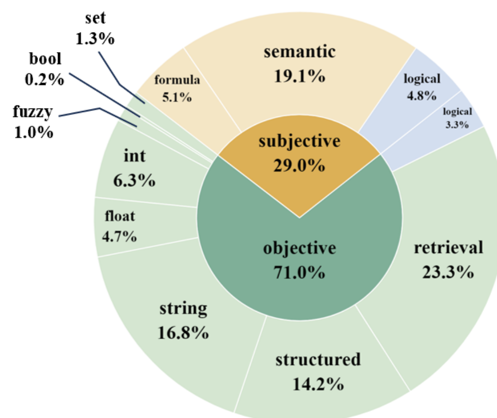

Figure 2: Distribution of different evaluation categories. 'bool', 'int', 'string', 'fuzzy' and 'structured' stand for specific evaluation functions in 'match' subtype.

Comparison with Existing Datasets In Table 3, we compare AirQA with existing scientific QA datasets. It is evident that our dataset demonstrates several salient strengths: 1) **More question types.** AirQA designs four different question types to systematically cover realistic research scenarios, 2) **More element types.** AirQA contains a wider variety of elements, including table, image, formula and metadata, and 3) **More precise evaluation.** AirQA employs 19 parameterized functions, which can be classified into two types and six subtypes, to facilitate customized evaluation.

Table 3: Comparison of our AirQA dataset and existing scientific QA datasets.

Dataset	**# QA**	**Evaluation Methods**	**Task types**				**Question based on**				
			Sgl.	Multi.	Retr.	Comp.	Full Text	Table	Image	Form.	Meta.
ScholarlyRead [30]	10K	BLEU, METEOR, ROUGE	✓	✗	✗	✗	✗	✗	✗	✗	✗
QASPER [9]	5,049	F1	✓	✗	✗	✗	✗	✗	✗	✗	✗
QASA [21]	1,798	Precision, Recall, F1, ROUGE	✓	✗	✗	✗	✓	✗	✗	✗	✗
SPIQA [28]	270K	METEOR, CIDEr, ROUGE, BERTScore, LLMLogScore	✓	✗	✗	✗	✓	✓	✓	✗	✗
PeerQA [5]	579	MRR, Recall, Rouge-L, AlignScore, Prometheus-2	✓	✗	✗	✗	✓	✗	✗	✓	✗
SciDQA [32]	2,937	ROUGE, BLEURT-20, BERTScore, LLM judge	✓	✓	✗	✗	✓	✓	✓	✓	✗
M3SciQA [22]	1,452	MRR, LLM judge	✗	✓	✗	✗	✓	✓	✓	✗	✗
AutoScholarQuery [13]	35K	Precision, Recall	✗	✗	✓	✗	✓	✗	✗	✗	✗
LitSearch [2]	597	Recall	✗	✗	✓	✗	✓	✗	✗	✗	✗
LitQA2 [33]	248	Precision, Accuracy	✗	✗	✗	✓	✓	✗	✗	✗	✗
AirQA (Ours)	1,246	Instance-level Function	✓	✓	✓	✓	✓	✓	✓	✓	✓

3 ExTrActor Framework for Trajectory Synthesis

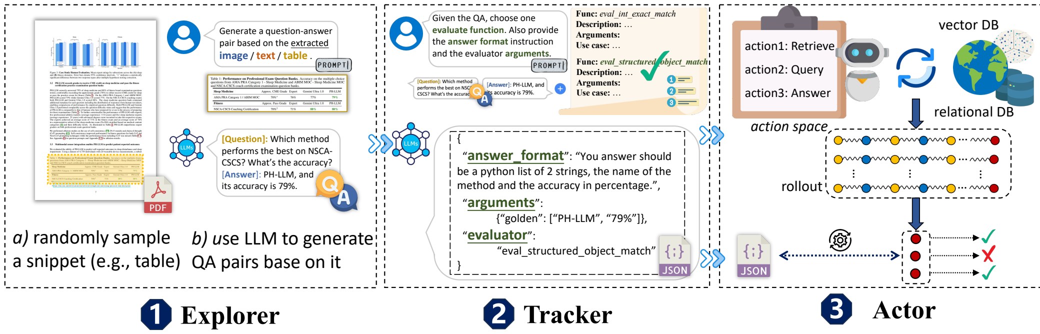

Figure 3: An overview of our ExTrActor framework, which consists of three stages to automatically extract QA pairs, formulate evaluation and answers, and filter valid agent trajectories.

In this section, we introduce our trajectory synthesis framework, ExTrActor, based on its three components: explorer, tracker and actor.

3.1 Overall Framework

To handle scientific QA, there are mainly three types of methods: 1) offering relevant information of the paper (e.g., title and abstract) alongside the question, 2) providing additional contexts via retrieval methods (e.g., RAG), and 3) equipping LLMs with supplementary tools, so that they can obtain sufficient information during multi-turn interactions. As discussed in Section 4.2, methods of the former two types significantly underperform the latter, even when supported by superior backbone models. Therefore, for the following fine-tuning, we apply Agentic Hybrid baseline (further illustrated in Section 4.1), whose environment includes a database and a vectorstore that produce execution or query results as observations when called by the agent with two predefined actions.

To mimic the real-world annotation and interaction scenarios, we split the synthesis process into three separate stages: 1) exploration stage, constructing a natural language question-answer pair with given context, 2) tracking stage, choosing suitable evaluation function and fill in the formatted example file, and 3) action stage, interacting with the outer environment to collect trajectories.

3.2 Explorer

Above all, we randomly download 10,000 papers in the artificial intelligence domain from `arXiv`, collect their metadata including titles and abstracts, and employ PyMuPDF [3] and `MinerU` [35], two tools designed for PDF parsing, to extract both textual and non-textual elements from the papers.

For the explorer, its goal is to generate rational question-answer pairs with sampled contexts. Based on different question types, we design three different modes: 1) For single type, we first randomly choose a paper and an element. Then, corresponding contexts are extracted according to the category of the element (e.g., `table_html` and `table_caption` for a table), and the explorer is expected to output a long-form question-answer pair. 2) Regarding retrieval type, instead of contexts, the explorer only receives the title and abstract as inputs, and is required to generate a question that indicates the paper. The corresponding answer is the title of that paper. 3) As for comprehensive type, we basically follow single type, while the only difference is that we provide the title and abstract along with the context, and ask the explorer to somehow elicit the paper that the question is about. To improve the quality of QA pairs, we adopt chain-of-thought [41] and hand-written category-based hint prompts.

3.3 Tracker

Regarding the tracker, its purpose is to wrap the previously generated natural language QA pairs into example files in accordance with specific formats. As explorer settings for different question types vary, we employ different tracker settings: 1) With regard to single and comprehensive questions, we provide the tracker with the QA pair along with the information of the evaluation functions, including descriptions, parameters and use cases. The tracker is then asked to choose the suitable evaluation function, fill in the parameters and the answer format, and refine the question-answer pair accordingly. 2) In terms of retrieval questions, as we restrict the answer to be the exact title of the chosen paper, we simply fill in the example file with fixed evaluation function, parameters and answer format. 3) As for multiple questions, the manual annotation consistently involves new papers outside our sampled collection, requiring real-time download and processing, which is incompatible with our explorer agent. For simplicity, we propose a rule-based combination of single examples: a) merging questions and answer formats with natural language templates, and b) combining evaluation functions with the logical function `evaluate_conjunction` (function details in App. A.3).

For more specific explorer and tracker prompts, please refer to App. E.

3.4 Actor

As for the actor, it aims at interacting with the environment to collect trajectories for instruction construction. In the outer environment, we include a database and a vectorstore containing corresponding information of the papers. Following AgentTuning [44], we employ ReAct [43] as the interaction framework, and design three actions to interact with the environment. For each synthetic example, we use a LLM as the actor to produce an interaction trajectory in a message list manner $(u_0, a_0, \ldots, u_i, a_i, \ldots, u_n, a_n)$, where u_i represents the user's instruction, or the observation from the environment, and a_i denotes the response from the agent LLM, including a thought and an action.

To avoid exceeding the context length of the models, we adopt the idea of sliding window and chunk the message list based on a window size of 5, generating multiple instruction data from one trajectory. During training, for each chunked list, we mask previous message history and train the last turn only.

We also observe that, some errors appear frequently in the collected trajectories (e.g., attempts to utilize undefined parameters). To ensure data quality, we remove instruction data that ends with a wrong action. For other instruction data, we reserve previous wrong actions and corresponding error information in the message list to guarantee error correction capability and coherence of thoughts.

4 Experiment

4.1 Agent Baselines

To comprehensively assess the LLMs on AirQA, we implement 8 baselines, including:

- **Trivial Baselines**: 1) Question Only baseline, only the question and the corresponding answer format are available, 2) Title-Abstract baseline, the titles and the abstracts of the corresponding papers are provided alongside, and 3) Full-Text with Cutoff baseline, raw textual contexts extracted from the papers are given in limited length.
- **Retrieval Baselines**: 1) RAG baseline, the question is sent to the vectorstore to retrieve relevant chunks, and LLMs answer the question based on retrieved contexts, and 2) Text2SQL baseline, where LLMs first generate a SQL, then answer the question based on the query results.
- **Agentic Baselines**: We employ a simple ReAct [43] framework with three actions: RETRIEVE, QUERY and ANSWER, corresponding to retrieving from the vectorstore, querying the database and generating the final answer. With this interactive framework, we implement 1) Agentic RAG and 2) Agentic Text2SQL baseline that only interact with the vectorstore and the database, respectively. And 3) Agentic Hybrid baseline with all actions utilized. Details on actions can be found in App. B.

Due to page limit, detailed experimental settings are all presented in Appendix C, along with an additional ablation that provides the agent with direct access to visual information to examine whether such input can further improve performance.

4.2 Main Results

Evaluation of Base Models To figure out different baselines' performance on the AirQA dataset, we choose two widely used models, GPT-4o and Qwen2.5-72B-Instruct, as representatives of proprietary and open-source LLMs. Table 4 shows that: 1) **Trivial baselines perform poorly.** Under Question Only setting, LLMs can only answer 5% of the questions correctly, demonstrating the quality of our

Table 4: Performance of different baselines on AirQA dataset.

Baseline	Question Type				Element Category					Evaluation		AVG
	sgl.	multi.	retr.	comp.	text	table	image	form.	meta.	obj.	subj.	
GPT-4o-2024-08-06												
Question Only	8.55	1.86	1.04	5.63	4.35	1.41	10.63	2.36	0.00	3.95	5.54	4.41
Title-Abstract	11.40	5.26	0.00	5.28	5.96	4.23	8.70	4.72	2.46	4.07	9.97	5.78
Full-Text w/ Cutoff	33.90	8.05	0.69	5.99	13.53	7.51	13.53	12.60	18.03	9.94	21.05	13.16
RAG	31.62	4.95	18.75	16.55	20.29	12.68	16.91	17.32	18.03	18.19	18.56	18.30
Text2SQL	21.08	6.81	7.64	17.25	14.01	8.92	12.08	16.54	14.75	11.41	18.28	13.40
Agentic RAG	34.19	8.36	15.63	29.58	21.36	18.78	26.57	22.83	24.59	21.36	24.10	22.15
Agentic Text2SQL	42.17	**11.15**	18.40	**38.38**	23.19	21.60	**28.99**	33.07	**47.54**	26.44	**31.02**	27.77
Agentic Hybrid	**45.58**	10.53	**52.13**	35.56	**39.61**	**23.00**	25.60	**33.86**	**47.54**	**38.76**	29.09	**35.96**
Qwen2.5-72B-Instruct												
Question Only	9.69	1.86	0.35	5.99	2.74	3.29	10.63	3.94	5.74	4.52	4.99	4.65
Title-Abstract	17.66	6.19	0.00	8.10	8.05	6.10	12.08	7.87	7.38	6.44	13.30	8.43
Full-Text w/ Cutoff	36.18	8.98	0.00	7.04	12.56	11.27	16.91	14.17	18.85	11.86	19.67	14.13
RAG	31.91	7.43	18.75	21.83	22.06	11.27	19.32	20.47	21.31	19.55	21.88	20.22
Text2SQL	22.22	4.02	11.11	13.38	13.85	8.45	15.46	10.24	11.48	12.43	14.13	12.92
Agentic RAG	32.76	9.60	15.63	30.28	22.06	15.96	25.12	25.98	18.85	21.02	25.21	22.23
Agentic Text2SQL	**43.02**	**11.46**	43.40	**40.14**	36.07	**21.13**	**29.95**	**35.43**	**49.18**	35.37	**31.59**	34.27
Agentic Hybrid	39.03	10.84	**55.21**	37.32	**41.71**	13.15	28.02	30.71	45.90	**37.74**	28.53	**35.07**

AirQA dataset. 2) **Provided more information sources, LLMs produce better answers.** With just a glimpse into the database or the vectorstore, retrieval baselines elevate the overall accuracy by at least 8% compared to Question Only baseline. 3) **As the interaction turn increases, LLMs explore the backend environment better.** While both agentic baselines outperform their retrieval counterparts, Agentic Text2SQL baseline exhibits significantly greater improvement, indicating that for structured retrieval like SQL, more searches enable step-by-step problem-solving, while for unstructured retrieval regarding the vectorstore, a single query is sufficient for most circumstances.

Table 5: Performance of Agentic Hybrid baseline with different backbone models on AirQA dataset.

Model	Question Type				Element Category					Evaluation		AVG
	sgl.	multi.	retr.	comp.	text	table	image	form.	meta.	obj.	subj.	
GPT-4o	45.58	10.53	52.13	35.56	39.61	23.00	25.60	**33.86**	47.54	38.76	29.09	35.96
o1-mini	37.04	12.07	45.14	24.65	35.43	14.55	22.22	22.83	36.07	31.07	26.04	29.61
Claude-3.7-Sonnet	45.30	15.17	58.68	27.46	43.96	22.07	24.64	27.56	44.26	39.32	29.64	36.52
Gemini-2.5-Pro	**51.85**	**18.58**	**67.01**	**40.49**	**51.53**	**29.58**	**29.95**	**33.86**	**53.28**	**46.55**	**38.23**	**44.14**
Qwen2.5-72B-Instruct	39.03	10.84	**55.21**	**37.32**	**41.71**	13.15	**28.02**	**30.71**	**45.90**	**37.74**	**28.53**	**35.07**
Llama-3.3-70B-Instruct	29.06	9.29	42.71	24.30	32.37	8.92	21.74	19.69	30.33	28.47	19.94	26.00
DeepSeek-R1	**41.03**	**11.46**	41.67	22.54	35.10	**15.96**	20.77	20.47	39.34	30.40	26.59	29.29

While baselines such as Question Only baseline exhibit a relatively low performance, Agentic Hybrid baseline consistently outperforms the others, allowing for comparisons between different models. In Table 5, we can observe that: 1) **Proprietary models outperform open-source models,** showing a stronger research capability, while some open-source models achieve performance comparable to closed-source ones. 2) **Reasoning models' performance on this method is not satisfactory,** possibly due to the incompatibility between their fixed reasoning formats and our interaction framework.

Evaluation of Fine-tuned Models For instrucion tuning, we choose the Qwen2.5 family [29] as base models. For all three agents in our ExTrActor framework, we employ Qwen2.5-32B-Instruct. While for the target model, unless otherwise specified, we utilize Qwen2.5-7B-Instruct as the backbone. We fine-tune three models, 3B, 7B and 14B, with instruction data extracted from the $4{,}000$ trajectories. Table 6 shows that, all three exhibit improved performance after training. While a reduction in the performance gain of 14B is observed, we consider it reasonable and acceptable. Please refer to Appendix D for a detailed analysis.

4.3 Ablation Study

Table 6: Performance of base/fine-tuned models of different scales on AirQA dataset. "FT" stands for fine-tuning, whether it is a base model, or a fine-tuned one.

Size	FT?	Question Type				Element Category					Evaluation		AVG
		sgl.	multi.	retr.	comp.	text	table	image	form.	meta.	obj.	subj.	
3B	✗	7.98	2.48	12.85	6.69	9.5	3.29	6.28	4.72	7.38	7.91	6.09	7.38
	✓	14.81	3.72	51.74	13.73	29.79	4.23	10.63	11.81	17.21	24.97	8.59	20.22
7B	✗	16.24	3.72	26.39	15.85	19.48	8.45	13.53	4.72	14.75	17.29	10.25	15.24
	✓	21.08	4.95	51.04	22.18	33.66	7.51	14.01	13.39	25.41	27.01	16.90	24.07
14B	✗	25.07	7.74	46.18	25.35	31.88	10.33	22.22	18.90	24.59	28.81	17.45	25.52
	✓	25.36	6.19	52.08	26.41	34.94	7.51	20.77	19.69	28.69	30.96	16.62	26.81
32B	✗	36.47	11.76	52.78	28.17	38.81	13.62	24.15	26.77	37.7	34.8	24.93	31.94

Component Ablation To evaluate the effect of the two proposed components, sliding window and error removal, we fine-tune two additional models: (1) one without either component, which uses raw trajectories as training data and computes loss over all model outputs, and (2) another that applies sliding window but retains all error actions. As shown in Table 7, sliding window yields a more pronounced improvement in the overall score, and a drastic reduction is also observed in error action rate, defined as $\frac{\text{\# error actions}}{\text{\# actions}}$, demonstrating the value of error removal in **improving the model's ability to generate valid actions**.

Table 7: Component Ablation.

Setting	Overall (%)	Error Rate(%)
base model	15.24	38.69
ExTrActor - w/o sliding window - w/o error removal	20.47	26.25
ExTrActor - w/o error removal	**24.08**	20.32
ExTrActor	24.07	**6.85**

Synthetic Data Scale We fine-tune another four models with more instruction data extracted from 1K, 2K, 4K and 10K trajectories. Figure 4 shows that, as the number of trajectory increase, the scores of all question types raise consistently, demonstrating the scalability of our method.

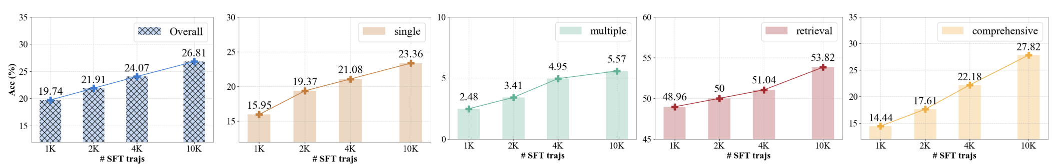

Figure 4: Performance changes with the increasing number of synthesized trajectories for fine-tuned Qwen-2.5-Instruct-7B model, split by different question types.

5 Related Work

PDF-based Scientific QA Datasets Previous research has consistently focused on the development of high-quality scientific QA datasets. Some datasets emphasize the accurate retrieval of papers based on specific descriptions [2, 13], while others concentrate on extracting detailed information from the text [5, 9, 21, 18, 30]. Recent work advance this field by incorporating non-textual elements such as table and image [28, 32]. M3SciQA [22] further extends this approach by introducing a paradigm for generating cross-document questions, while LitQA2 [33] pay more attention to the combination of paper retrieval and detailed QA. Our AirQA dataset innovates in this field by encompassing various question types and element categories, while designing a function-based instance-level evaluation.

Instruction Tuning and Synthetic Data Instruction tuning serves as a useful tool for aligning LLMs with human instructions, but it requires corresponding outputs for specific instructions, thus heavily relying on high-quality training data [10, 23, 42]. While manual annotation is an effective method, it is hard to scale up due to time and cost constraints. Previous work, such as SciQAG [34], Self-Instruct [40] and BUTTON [7], introduced different methods for crafting synthetic instruction data from scratch, while AgentTuning [44] proposed extracting data from interaction trajectories to leverage existing datasets. In this work, we introduce a multi-agent framework, ExTrActor, to automate both example generation and trajectory collection, facilitating instruction data synthesis.

6 Conclusion

In this work, we manually annotate a multi-modal multi-task multi-paper dataset (AirQA) with instance-level evaluation, and a multi-agent framework (ExTrActor) for instruction data synthesis. Evaluations demonstrate the quality of our dataset, indicating the challenges current models face in scientific QA, while experiments on instruction tuning highlight the effectiveness of the framework. Future works include: 1) exploring other RL-based methods for further improvements, and 2) extend the PDF-based tasks to other knowledge intensive domains, including law and medicine.

Acknowledgments and Disclosure of Funding

We would like to thank Haoran Wang, Jingyi Zhang, Ye Wang, Yuxun Miao, Danyang Zhang, Hanqi Li, Zichen Zhu, Situo Zhang, Senyu Han, Dingye Liu, Wenjie Sun, Hanchong Zhang, Nan Jiang, Liangtai Sun, Da Ma, Hankun Wang, and Zhihan Li for their careful annotation on the AirQA dataset.

References

[1] Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges, 2024.

[2] Anirudh Ajith, Mengzhou Xia, Alexis Chevalier, Tanya Goyal, Danqi Chen, and Tianyu Gao. Litsearch: A retrieval benchmark for scientific literature search, 2024.

[3] Inc. Artifex Software. Pymupdf - a python binding for mupdf. `https://pymupdf.readthedocs.io/en/latest/`, 2023. Version 1.24.9, accessed on January 25, 2025.

[4] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025.

[5] Tim Baumgärtner, Ted Briscoe, and Iryna Gurevych. PeerQA: A scientific question answering dataset from peer reviews. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 508–544, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[6] 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. Improving language models by retrieving from trillions of tokens, 2022.

[7] Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. Facilitating multi-turn function calling for llms via compositional instruction tuning, 2025.

[8] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[9] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers, 2021.

[10] Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. How abilities in large language models are affected by supervised fine-tuning data composition, 2024.

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

[12] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020.

[13] Yichen He, Guanhua Huang, Peiyuan Feng, Yuan Lin, Yuchen Zhang, Hang Li, and Weinan E. Pasa: An llm agent for comprehensive academic paper search, 2025.

[14] Xanh Ho, Jiahao Huang, Florian Boudin, and Akiko Aizawa. Llm-as-a-judge: Reassessing the performance of llms in extractive qa, 2025.

[15] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey, 2024.

[16] Hugging Face Team. Llama 3.3-70b-instruct model. `https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct`, 2023. Accessed: 2023-02-11.

[17] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models, 2022.

[18] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. PubMedQA: A dataset for biomedical research question answering. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2567–2577, Hong Kong, China, November 2019. Association for Computational Linguistics.

[19] Ehsan Kamalloo, Shivani Upadhyay, and Jimmy Lin. Towards robust qa evaluation via open llms. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2811–2816, New York, NY, USA, 2024. Association for Computing Machinery.

[20] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[21] Yoonjoo Lee, Kyungjae Lee, Sunghyun Park, Dasol Hwang, Jaehyeon Kim, Hong-In Lee, and Moontae Lee. QASA: advanced question answering on scientific articles. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19036–19052. PMLR, 2023.

[22] Chuhan Li, Ziyao Shangguan, Yilun Zhao, Deyuan Li, Yixin Liu, and Arman Cohan. M3sciqa: A multi-modal multi-document scientific qa benchmark for evaluating foundation models, 2024.

[23] Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning, 2024.

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[25] Hannes Mühleisen and Mark Raasveldt. *duckdb: DBI Package for the DuckDB Database Management System*, 2024. R package version 1.1.3.9017, https://github.com/duckdb/duckdb-r.

[26] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022.

[27] Azizi Othman. Manus ai: Capabilities, limitations, and market position, 03 2025.

[28] Shraman Pramanick, Rama Chellappa, and Subhashini Venugopalan. Spiqa: A dataset for multimodal question answering on scientific papers, 2025.

[29] 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, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[30] Tanik Saikh, Asif Ekbal, and Pushpak Bhattacharyya. ScholarlyRead: A new dataset for scientific article reading comprehension. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5498–5504, Marseille, France, May 2020. European Language Resources Association.

[31] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[32] Shruti Singh, Nandan Sarkar, and Arman Cohan. Scidqa: A deep reading comprehension dataset over scientific papers, 2024.

[33] Michael D. Skarlinski, Sam Cox, Jon M. Laurent, James D. Braza, Michaela Hinks, Michael J. Hammerling, Manvitha Ponnapati, Samuel G. Rodriques, and Andrew D. White. Language agents achieve superhuman synthesis of scientific knowledge, 2024.

[34] Yuwei Wan, Yixuan Liu, Aswathy Ajith, Clara Grazian, Bram Hoex, Wenjie Zhang, Chunyu Kit, Tong Xie, and Ian Foster. Sciqag: A framework for auto-generated science question answering dataset with fine-grained evaluation, 2024.

[35] Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, Bo Zhang, Liqun Wei, Zhihao Sui, Wei Li, Botian Shi, Yu Qiao, Dahua Lin, and Conghui He. Mineru: An open-source solution for precise document content extraction, 2024.

[36] Cunxiang Wang, Sirui Cheng, Qipeng Guo, Yuanhao Yue, Bowen Ding, Zhikun Xu, Yidong Wang, Xiangkun Hu, Zheng Zhang, and Yue Zhang. Evaluating open-QA evaluation. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[37] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2614–2627, 2021.

[38] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models, 2023.

[39] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[40] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023.

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[42] Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. Baize: An open-source chat model with parameter-efficient tuning on self-chat data, 2023.

[43] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[44] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms, 2023.

[45] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics.

A AirQA Dataset

A.1 Example Format

Each data instance in AirQA is represented as a JSON dictionary containing the following fields:

- `uuid`: Globally unique uuid of the current task example.
- `question`: The user's question about the given papers.
- `answer_format`: The requirements for the output format of LLMs, e.g., "a Python list of text strings" or "a single float number", so that LLMs can form their answer accordingly and we can evaluate the answer conveniently.
- `tags`: A list of tags denoting different question types, element categories and evaluation types. Feasible tags include ["single", "multiple", "retrieval", "text", "image", "table", "formula", "metadata", "subjective", "objective"], corresponding to question types, modal categories in Section 2.1 and evaluation types in Section 2.2.
- `anchor_pdf`: A list of PDF uuids that are directly related to or explicitly mentioned in the question, provided in single and multiple questions.
- `reference_pdf`: A list of PDF uuids that may or may not help answer the question, only provided in multiple questions.
- `conference`: A list of conference name followed by year, provided in retrieval and comprehensive questions.
- `evaluator`: A dictionary containing 2 fields, `eval_func` and `eval_kwargs`, which defines how to evaluate the model outputs. Concretely,
  - the "`eval_func`" field defines the name of our customized Python function (or metric) which is used to compare the predicted result and the expected ground truth;
  - the "`eval_kwargs`" field defines the arguments for the corresponding evaluation function, which usually contain the gold or reference answer and other optional parameters.

A.2 Metadata Format

The metadata of each paper is organized in a structured JSON format, capturing key bibliographic and content-related attributes, as shown below:

- `uuid`: A universally unique identifier (UUID) assigned to this specific data sample.
- `title`: The full title of the academic paper.

Table 8: The checklist of the 19 used evaluation functions, including their genres, categories, names, and descriptions.

Genre	Category	Function	Description
objective	match	eval_bool_exact_match	Evaluate the output against the answer using exact boolean match.
		eval_float_exact_match	Evaluate the output against the answer using exact float match with variable precision or tolerance.
		eval_int_exact_match	Evaluate the output against the answer using exact integer match.
		eval_string_exact_match	Evaluate the output against the answer using exact string match.
		eval_string_fuzzy_match	Evaluate the output against the answer using fuzzy match provided by FuzzyWuzzy.
		eval_structured_object _exact_match	Evaluate the output against the answer recursively by parsing them both as Python-style lists or dictionaries.
	set	eval_element_included	Evaluate whether the output is included in the answer list.
		eval_element_list_included	Evaluate whether each element in the output list is included in the answer list.
		eval_element_list_overlap	Evaluate whether the output list overlaps with the answer list.
	retrieval	eval_paper_relevance_with _reference_answer	Evaluate whether the retrieved paper is the same as the reference answer.
subjective	semantic	eval_reference_answer _with_llm	Evaluate the output against the reference answer using LLMs.
		eval_candidate_reference _answer_with_llm	Evaluate whether the output matches any candidate reference answer.
		eval_scoring_points_with _llm	Evaluate whether the scoring points are all mentioned in the output using LLMs.
		eval_partial_scoring_points _with_llm	Evaluate whether the scoring points are partially mentioned in the output using LLMs.
		eval_reference_answer_and _scoring_points_with_llm	Evaluate whether the reference answer and other scoring points are all mentioned in the output using LLMs.
	formula	eval_complex_math_form ula_with_llm	Evaluate the mathematical equivalence between the output and the answer formatted in Latex using LLMs.
logical		eval_conjunction	Evaluate the conjunction of multiple evaluation functions. The output passes the evaluation if and only if all the elements in the output pass the corresponding sub-evaluations.
		eval_disjunction	Evaluate the disjunction of multiple evaluation functions. The output passes the evaluation if and only if at least one of the element in the output passes the corresponding sub-evaluation.
		eval_negation	Evaluate the negation of an evaluation function. The output passes the evaluation if and only if it doesn't pass the original evaluation function.

- `conference_full`: The complete official name of the conference where the paper was published or presented. For example, "Annual Meeting of the Association for Computational Linguistics".
- `conference`: The standardized or abbreviated name of the conference, commonly used in citations or file naming. Examples include "ACL", "NeurIPS".
- `year`: The year in which the paper was published or presented at the conference.
- `volume`: The volume information of the conference proceedings or journal in which the paper appears, if applicable. For example, "NeurIPS 2023 poster".
- `bibtex`: A string containing the full BibTeX citation entry for the paper.
- `authors`: An ordered list of the authors who contributed to the paper.
- `pdf_url`: A direct URL linking to the downloadable PDF file of the paper. The link should point to an actual PDF file and therefore must end with ".pdf".
- `pdf_path`: The local file system path where the PDF is saved. The file should be renamed using the UUID to ensure consistent and collision-free naming.
- `num_pages`: An integer value indicating the total number of pages in the PDF document.
- `abstract`: The abstract of the paper, which is a concise summary of the research objectives, methodology, key findings, and implications.
- `tldr`: "Too Long; Didn't Read" summary — a brief, high-level summary of the paper's main contribution, typically one to two sentences.
- `tags`: A list of keywords or topic tags associated with the paper.

A.3 Evaluation Function

In Table 8, we list the detailed names and descriptions of all 19 evaluation functions.

Here we present three representative cases corresponding to three distinct categories of evaluation functions: objective functions, subjective functions, and logical functions.

```json
{
    "example": {
        "eval_func": "eval_string_exact_match",
        "eval_kwargs": {
            "gold": "Italian",
            "lowercase": true
        }
    }
}
```

Listing 1: Objective Function Case

In case (Listing 1), the evaluation function compares the predicted answer with the gold answer 'Italian', ignoring case sensitivity due to the 'lowercase' parameter being set to true. This means 'italian', 'ITALIAN', or 'ItAliAn' would all match 'Italian'.

```json
{
    "example": {
        "eval_func": "eval_reference_answer_with_llm",
        "eval_kwargs": {
            "reference_answer": "Artificial intelligence is a branch
                ↪ of computer science focused on building systems
                ↪ that can perform tasks that typically require
                ↪ human intelligence.",
            "question": "What is artificial intelligence?"
        }
    }
}
```

Listing 2: Subjective Function Case

In case (Listing 2), the evaluation checks if the predicted answer conveys the same meaning as the reference answer about artificial intelligence, in response to the question 'What is artificial intelligence?'.

The prompt used for conducting the subjective evaluation above is presented as an example among several possible formulations within this evaluation category, and is shown below.

Subjective Function Prompt Example

You are an intelligent judgement system who is expert in determining whether a predicted answer matches the reference answer in terms of semantic meaning and intent, based on the input question. You will be given the raw question, the reference answer, and the predicted answer. And you need to provide the final decision with the following format:
```txt
True/False
```
Notice that:
1. Remember to wrap the final judgement with triple backticks.
2. The final decision string must exactly be "True" or "False" without any extra character or punctuation. Any other text will be considered as incorrect.
3. The structure and format of the predicted answer do not matter. We only care about the semantic content, compared to the reference answer. Minor differences in grammar, structure, or formatting should be ignored if the core meaning is preserved.
Now, let's start!

[Question]: {question}
[Reference Answer]: {reference_answer}
[Predicted Answer]: {predicted_answer}

Let's think step-by-step, and then provide the final judgement.

It is worth noting that: 1) While LLM-based judgment is necessary in certain cases, we design **tailored prompts to support more fine-grained and targeted evaluation** unlike previous datasets which mostly rely on a fixed prompt. e.g., we design a specialized prompt to compare LaTeX formulas. 2) Existing studies also suggest that, for QA tasks, LLM-based evaluations are more aligned with human judgments than metrics such as accuracy or F1 [36, 14, 19]. 3) Analysis on 66 examples shows that LLM-based and human evaluations are largely consistent, with an agreement rate of approximately 83%.

```
{
    "example": {
        "eval_func": "eval_disjunction",
        "eval_kwargs": {
            "eval_func_list": [
                "eval_string_exact_match",
                "eval_reference_answer_with_llm"
            ],
            "eval_kwargs_list": [
                {
                    "gold": "role-oriented routing",
                    "lowercase": true
                },
                {
                    "reference_answer": "It routes messages,
                       ↪ requests, or tasks based on the roles or
                       ↪ responsibilities of the recipients, rather
                       ↪ than simply by their identity or static
                       ↪ attributes.",
                    "question": "What's the most important idea of
                       ↪ role-oriented routing?"
                }
            ]
        }
    }
}
```

Listing 3: Logical Function Case

In case (Listing 3), the disjunction evaluation function checks if at least one of the specified evaluation functions returns a positive result. The first function, ‘eval_string_exact_match‘, verifies whether the predicted answer matches the gold standard ’role-oriented routing’ in a case-insensitive manner. The second function, ‘eval_reference_answer_with_llm‘, evaluates whether the predicted answer sufficiently addresses the question about the most important idea of role-oriented routing, as described in the provided reference answer. If either condition is satisfied, the evaluation returns 1.0; otherwise, it returns 0.0.

A.4 Additional Dataset Statistics

Here we include another two statistics on paper volume in Figure 2 and question lengths in Figure 6.

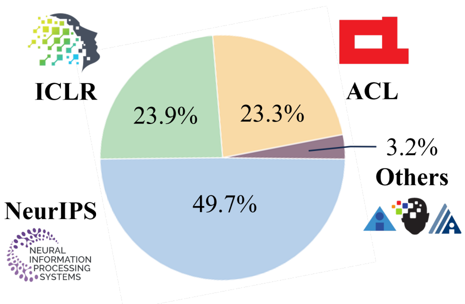

Figure 5: Conference distribution of the papers used.

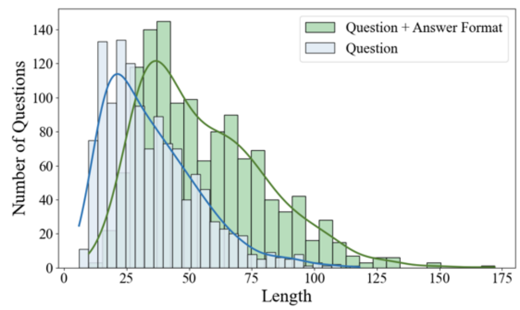

Figure 6: Distribution of question and answer format lengths (in tokens) by count.

B Agentic Baseline

In this section, we present detailed information of the actions implemented in the agentic baselines. Each action can be called in a Python-style manner, e.g., `Retrieve(query=‘‘Is there any work about the topic structured RAG?’’, limit=4)`

```
{
    "action_type": "Retrieve",
    "description": "Given a query text, retrieve relevant context
        ↪ from the Milvus vectorstore.",
    "observation": "The observation space is the retrieved top-ranked
        ↪ entries from the Milvus vectorstore based on queries.",
    "parameters": {
        "query": {
            "type": "str",
            "required": true,
            "description": "The query text will be encoded and used
                ↪ to search for relevant context. You can rephrase
                ↪ the original user question to obtain a more clear
                ↪ and structured query requirement."
        },
        "limit": {
            "type": "int",
            "required": false,
            "default": 5,
            "description": "The number of top-ranked context to
                ↪ retrieve. Please set it to a positive integer to
                ↪ limit the number of returned results. Extremely
                ↪ large limit values may be truncated."
        }
    },
```

```
    "use_cases": [
        {
            "example": {
                "query": "Is there any work about the topic
                    ↪ structured RAG?"
            },
            "explanation": "Retrieve top 5 pieces about a certain
                ↪ topic."
        },
        {
            "example": {
                "query": "What's the learning rate for training the
                    ↪ ResNet model?",
                "limit": 4
            },
            "explanation": "Retrieve detailed information about the
                ↪ learning rate for training the ResNet model. The
                ↪ top 4 most relevant entries will be returned based
                ↪ on the query."
        }
    ]
},
{
    "action_type": "Query",
    "description": "Generate an SQL query to retrieve the desired
        ↪ information from the DuckDB database. Please refer to the
        ↪ concrete database schema to produce a valid and executable
        ↪ SQL.",
    "observation": "The observation space is the execution result of
        ↪ the SQL query. You do not need to worry about the actual
        ↪ execution, we will perform it for you. If the SQL failed
        ↪ to execute, we will return the error message. Extremely
        ↪ long SQL output will be truncated.",
    "parameters": {
        "sql": {
            "type": "str",
            "required": true,
            "description": "The concrete DuckDB SQL query to execute
                ↪ and retrieve results."
        }
    },
    "use_cases": [
        {
            "example": {
                "sql": "SELECT abstract FROM metadata WHERE paper_id
                    ↪ = '12345678';"
            },
            "explanation": "Get the abstract of the paper with
                ↪ paper_id '12345678' from the metadata table in the
                ↪ DuckDB database."
        },
        {
            "example": {
                "sql": "SELECT pages.page_number FROM images JOIN
                    ↪ pages JOIN metadata ON images.ref_page_id =
                    ↪ pages.page_id AND pages.ref_paper_id =
                    ↪ metadata.paper_id WHERE metadata.paper_id =
                    ↪ '12345678' AND images.image_caption LIKE
                    ↪ '%Figure 3%';"
            },
            "explanation": "Find which page in the paper with
                ↪ paper_id '12345678' contains Figure 3."
        }
    ]
},
```

```
{
    "action_type": "Answer",
    "description": "When you take this action, the retrieved results
        ↪ suffice to answer the user question. PLEASE STRICTLY
        ↪ ADHERE TO THE ANSWER FORMAT FOR THE CURRENT QUESTION.",
    "observation": "There is no observation for this terminal action,
        ↪ since it indicates the completion of the task and end of
        ↪ the interaction.",
    "parameters": {
        "answer": {
            "type": "Any",
            "required": true,
            "description": "The final answer to the user question."
        }
    },
    "use_case": [
        {
            "example": {
                "answer": 42
            },
            "explanation": "The final answer is 42."
        },
        {
            "example": {
                "answer": ["Results", "Discussion"]
            },
            "explanation": "The final answer is a list of strings:
                ↪ ['Results', 'Discussion']."
        }
    ]

}
```

Listing 4: Detailed Action Format

C Supplementary Experiments and Settings

C.1 Preprocessing

Besides collecting papers and metadata illustrated in Section 2.3, we parse the papers with PyMuPDF [3] and `MinerU` [35], and populate relevant information into the relational database DuckDB [25]. Additionally, we segment raw documents into chunks of 512 tokens, encode the chunks with `all-MiniLM-L6-v2` [39], and insert the vectors into the vectorstore Milvus [37].

C.2 LLMs and Hyper-Parameters

We evaluate various LLMs on AirQA. For closed-source ones, we use GPT-4o-2024-08-06, o1-mini-2024-09-12, Claude-3.7-Sonnet-20250219 and Gemini-2.5-Pro-exp-03-25. Regarding open-source LLMs, we employ Qwen2.5-72B-Instruct [29], Llama-3.3-70B-Instruct [16], and DeepSeek-R1 [11]. As for hyper-parameters, the $\mathrm{temperature}$ is set to 0.7 and $\mathrm{top\_p}$ is fixed to 0.95. Specifically, for reasoning models, the $\mathrm{temperature}$ is set to 0.6. The maximum retrieved tokens in each turn and the cutoff for full-text input are both limited to $5\mathrm{K}$. The threshold of interaction turns is 20 and the window size for the message history is 5. For closed-source models, we directly call their API serviecs, while for open-source ones, we deploy them on NVIDIA A800 Tensor Core clusters using vLLM [1] [20].

C.3 Instruction Tuning

For instrucion tuning, we choose the Qwen2.5 family [29] as base models. For all three agents in our ExTrActor framework, we employ Qwen2.5-32B-Instruct. While for the target model, unless

[1] `https://docs.vllm.ai/en/latest/index.html`

Table 9: Hyper-parameters for Instruction Tuning.

Hyper-Parameter	**Default Value**
Finetuning Type	LoRA
LoRA Target	all
LoRA Rank	16
LoRA Alpha	16
LoRA Dropout	0.05
Cutoff Length	4,096
Mask History	true
Gradient Accumulation Steps	16
Learning Rate	1×10^{-4}
Train Epochs	1.0
Learning Rate Scheduler	Cosine
Warmup Ratio	0.1

otherwise specified, we utilize Qwen2.5-7B-Instruct as the backbone. As for the training framework, we employ LLaMA-Factory [45]. Regarding our synthetic instruction data, we transform them into standardized ShareGPT foramt following Vicuna [8], and as mentioned before, we only compute the loss of the model's last output during fine-tuning by setting `mask_history` as true. By default, we use a learning rate of 1×10^{-4}, apply AdamW optimizer [24] with a cosine learning scheduler and train for one epoch. The entire training process is conducted on two Ascend 910B4 NPUs with 64GB of memory each.

The detailed hyper-parameters used in LLaMA-Factory for instruction tuning are listed in Table 9.

C.4 Ablation on VLMs

To further investigate whether direct access to visual information improves performance, we implement an additional action, "View", for the Agentic Hybrid baseline.

```
{
    "action_type": "View",
    "description": "You can retrieve the visual information of the
        ↪ paper by taking this action. Please provide the paper id,
        ↪ the page number, and the optional bounding box.",
    "observation": "The observation space is the image that you want
        ↪ to view. We will show you the image according to your
        ↪ parameters. The error message will be shown if there is
        ↪ any problem with the image retrieval.",
    "parameters": {
        "paper_id": {
            "type": "str",
            "required": true,
            "description": "The paper id to retrieve the image."
        },
        "page_number": {
            "type": "int",
            "required": true,
            "description": "The page number (starting from 1) of the
                ↪ paper to retrieve the image."
        },
        "bounding_box": {
            "type": "List[float]",
            "required": false,
            "default": [],
            "description": "The bounding box of the image to
                ↪ retrieve. The format is [x_min, y_min, delta_x,
```

```
                ↪ delta_y]. The complete page will be retrieved if
                ↪ not provided."
            }
        },
        "use_cases": [
            {
                "example": {
                    "paper_id": "12345678",
                    "page_number": 3,
                    "bounding_box": []
                },
                "explanation": "Retrieve the image of the third page of
                ↪ the paper with id 12345678."
            },
            {
                "example": {
                    "paper_id": "12345678",
                    "page_number": 5,
                    "bounding_box": [
                        51.1,
                        204.3,
                        333.0,
                        13.8
                    ]
                },
                "explanation": "Retrieve the image of the fifth page of
                ↪ the paper with id 12345678, with a bounding box of
                ↪ [51.1, 204.3, 384.1, 218.1]."
            }
        ]
}
```

Listing 5: Details for "View" action.

Through this action, LLMs can **access the image content encoded in base64 format** by specifying the paper ID, page number, and the bounding box of the relevant region.

Table 10: Performance of Agentic Hybrid with "View" action on AirQA dataset.

Size	**"View"?**	**text**	**table**	**image**	**form.**	**meta.**	**AVG**
32B	✗	31.88	10.33	17.87	18.90	21.31	24.24
	✓	31.56	8.92	20.77	15.75	25.41	24.56
72B	✗	36.55	15.96	23.67	20.47	36.88	30.34
	✓	36.55	15.96	24.64	22.83	36.07	30.78

We conduct experiments with Qwen2.5-VL family [4] as backbone model. Table 10 shows that, with direct access to images, the performance on visual questions improves, though further methods for enhancement remain to be explored.

C.5 Fine-tuning based on AirQA

To examine the relative effectiveness of synthetic data compared to manually annotated data, we fine-tune another model with examples directly from the AirQA dataset. Due to the unquantifiable nature of manually annotated trajectories, we continue to use Qwen2.5-32B-Instruct as teacher model for trajectory generation.

Table 11 indicates that, model fine-tuned on 400 examples from AirQA achieves performance comparable to that trained on 1,000 automated generated examples. It is worth noticing that while manual examples seem to be more effective, they come at a higher cost. It takes 20 minutes for a human to generate an example and only 20 seconds for EXTRACTOR.

Table 11: Comparison between models fine-tuned on examples directly from the AirQA dataset and examples generated by the EXTRACTOR.

Dataset	Count	sgl.	multi.	retr.	comp.	AVG
-	-	16.24	3.72	26.39	15.85	15.24
AirQA	400	15.67	4.95	48.26	13.73	19.98
EXTRACTOR	1000	15.95	2.48	48.96	14.44	19.74

C.6 Human Study

To provide a reference for the difficulty of the AirQA dataset, we recruit 3 students with expertise in artificial intelligence to answer 98 questions sampled from our AirQA dataset. They are strictly prohibited from using any form of LLM and are only allowed to search the internet. Each question has a time limit of 20 minutes.

Table 12: Performance of human experts on AirQA dataset.

Question Type				Element Category					Evaluation		AVG
sgl.	multi.	retr.	comp.	text	table	image	form.	meta.	obj.	subj.	
64.29	54.00	52.17	56.82	53.12	56.07	67.05	50.00	55.00	58.52	52.58	56.63

Results in table 12 show that AirQA is a highly challenging dataset, even human experts are only able to achieve a relatively high score within the time limit, rather than a perfect one.

C.7 Statistcs on Time & Cost

In Table 13 we list the time and cost per example for two models and three agentic baselines for reference.

Table 13: Statistics of the number of interaction(s), accumulated prompt / completion token(s), time consumption, and LLM cost per sample with different models and agentic methods on AirQA.

Model	RAG Method	# Turn(s)	# Prompt Token(s)	# Completion Token(s)	Time (s)	Cost ($)
Qwen2.5-72B-instruct	Agentic RAG	7.53	39870	658	58	-
	Agentic Text2SQL	6.45	42991	790	70	-
	Agentic Hybrid	5.95	62533	767	62	-
GPT-4o	Agentic RAG	4.59	13231	365	13	0.0367
	Agentic Text2SQL	7.26	32957	815	22	0.0905
	Agentic Hybrid	5.08	35909	566	18	0.0954

D Supplementary Analysis

D.1 Reduction in the Performance Gain of Fine-tuned 14B

We regard this reduction in performance gain as fully reasonable and acceptable, because: 1) The actor agent in our EXTRACTOR framework in effect serves as a teacher agent. The selected teacher model Qwen2.5-32B-Instruct scores only 31.94% on AirQA, which represents the **upper performance bound achievable through distillation**. 2) While this diminishing return has long been a meaningful and widely discussed research question beyond the scope of this paper, we can observe that with EXTRACTOR, **small models produce significantly less errors in predicting actions**. As shown in Table 7, 7B's error action rate drops from 38.69% to 6.85%, and similar improvement is observed on 14B, with error action rate dropping from 31.63% to 6.64%, indicating the effectiveness of our framework.

D.2 Error Analysis on GPT-4o

To further illustrate the bottleneck of our AirQA dataset, we randomly sample 60 examples (15 for each question type) where GPT-4o + Agentic Hybrid produces incorrect answers. Through manual analysis, we identify the following five root causes that ultimately lead to mistakes:

1. **Lack of Context**: The agent fails to use the given tools to find relevant snippets.
2. **Over Confidence**: The agent chooses to generate the answer too early.
3. **Missing Paper**: For questions that involve multiple papers, the agent fails to realize/find other papers.
4. **Textual Reasoning**: The agent successfully retrieves the key snippet but fails to understand it.
5. **Visual Reasoning**: The agent fails to retrieve/understand paratextual information.

Table 14: Error Analysis of Agentic Hybrid baseline with GPT-4o as backbone.

Category	sgl.	multi.	retr.	comp.	Total
Lack of Context	**5**	3	**9**	**6**	**23**
Over Confidence	3	2	5	3	13
Textual Reasoning	2	1	1	5	9
Missing Paper	0	**8**	0	0	8
Visual Reasoning	**5**	1	0	1	7

The analysis shows that the dominant cause differs across question types, while generally, current models still exhibit limitations in **long-term planning and multi-modal reasoning**, suggesting the need for new methods that better balance planning and acting.

Table 15: Error Analysis of Agentic Hybrid baseline with fine-tuned 7B as backbone.

Category	sgl.	multi.	retr.	comp.	Total
Lack of Context	4	**4**	**9**	3	**20**
Textual Reasoning	**6**	2	3	2	13
Over Confidence	2	3	2	3	10
Repetition	1	3	1	**4**	9
Visual Reasoning	2	1	0	3	6
Missing Paper	0	2	0	0	2

D.3 Error Analysis on Fine-tuned 7B

Similarly, we conduct an error analysis on fine-tuned Qwen2.5-7B-Instruct. Besides the aforementioned five causes, we identify another reason that ultimately leads to failures:

6. **Repetition**: The agent consistently predicts the same action.

The analysis indicates that fine-tuned models still exhibit limitations in **textual retrieval and comprehension**, highlighting areas for future improvement.

E Synthesis Prompt

E.1 Explorer Prompt

Here we showcase prompt templates of explorers discussed in Section 3.2.

Explorer Prompt for Single and Comprehensive question types

````
You are an intelligent annotation system who is expert in posing questions.
{description}
Your output should be in the following format:
[Thought]: Your thought process.
```txt
[Question]: Your question here.
[Answer]: Your answer here.
```
Notice that:
- Remember to wrap your output (except [Thought]) with triple backticks.
- Don't include the answer in the question or in the reasoning steps.
- Your question should be as objective as possible.
- Your answer should be concise and clear.
{hint}
Let's think step-by-step, and then provide the final question and answer.
{context}
````

Here, `{description}` stands for the description of the task, `{hint}` includes additional hints, and `{context}` represents the corresponding context for question generation.

For example, for single question type and table element category, the description is *"You will be given an AI research paper, and your task is to generate a question based on the content of the table in HTML format and the caption of the table."*, the hint prompt is *"- Try not to include the word 'table' in your question."*, and the context prompt is:

Context Prompt for Single type, Table category

````
The caption of the table is as follows:
```txt
{caption}
```
The content of the table is as follows:
```html
{content}
```
````

where the caption and the content are the raw text caption and the table content in HTML format respectively.

Explorer Prompt for Retrival question type

````
You are an intelligent annotation system who is expert in posing questions. You need to pose a question based on the title and abstract of a paper, where the answer to the question should be the title of the paper. That is to say, you need to describte the contribution or the feature of the paper in the question, so that the respondents can identify the paper. Don't include the title itself in the question. Now let's start!

[Title]: {title}
[Abstract]: {abstract}

Your output should be in the following format:

Your thought process.
```txt
Your question here.
```

Note that, you should wrap your output with triple backticks.
````

For retrieval question type, we use the different explorer prompt shown above, and for multiple question type, there is no need of an explorer, as we simply combine two single questions.

E.2 Tracker Prompt

Here we present prompt templates of trackers discussed in Section 3.3.

Tracker Prompt for Single and Comprehensive question types

You are an intelligent annotation system who is expert in reviewing questions.

You will be given a question and an answer. You should adjust the question and the answer, adapting them to the evaluator's requirements. The descriptions, parameters and use cases of the evaluators are provided below:

{evaluator}

Note that:
- If you want the predicted answer list to be exactly same with the gold answer list, use `eval_structured_object_exact_match`, don't use `eval_element_list_included`.
- If your evaluation involves list matching, and the order doesn't matter, set `ignore_order` to `true`. If the order matters, set `ignore_order` to `false`.
- If you are sure that the answer is unique, there aren't other equivalent answers, and any rephrase will change the semantic meaning of the answer, you can use `eval_string_exact_match`. Otherwise, you should use `eval_reference_answer_with_llm`. Generally, we recommend using `eval_reference_answer_with_llm` for subjective questions, and `eval_string_exact_match` for single-word answers.

Your output should be in the following format:
[thought]: Your thought process.
```txt
[question]: Modified question.
[evaluator]: The evaluator you choose.
[answer_format]: The format that the respondent should follow in order to pass the evaluator. e.g. "Your answer should be a single python list containing two strings, the first element of the list is the abbreviation of the baseline, the second element of the list is the full name of this baseline, e.g.["abbr","full"].".
[answer]: Modified answer.
[tag]: A single `subjective` or `objective` without explanation. Whether the evaluator involves LLM. `subjective` if it involves LLM, otherwise `objective`.
```

Note that:
- Remember to wrap your output (except thought) with triple backticks.
- DON'T INCLUDE ANSWERS, HINTS OR KEY POINTS IN [question] OR [answer_format] IN ANY FORM, ESPECIALLY WHEN YOU TRY TO ILLUSTRATE [answer_format] BY GIVING EXAMPLES.
- [answer_format] will be provided to the respondent along with the [question]. [question] and [answer_format] together form the who question that will be presented to the respondent. [question] focuses on the question itself, [answer_format] focuses on the format of the answer.
- You should present [evaluator] in JSON format, as given in the use cases. And your [answer] should be able to pass the evaluator.
- You can modify the question and answer based on the evaluator's requirements, but don't change the original meaning of the question and answer.
- When the question involves percentage, and the percentage is an exact value, not an approximate value, try to use `eval_float_exact_match` or `eval_int_exact_match`, while indicating the decimal places in [answer_format].

Here're the original question and answer:
```txt
[question]: {question}
[answer]: {answer}
```

Let's think step-by-step, and then provide the final arguments.

Here, {evaluator} contains the detailed information of the 19 evaluation functions, and {question} and {answer} stand for the question-answer pair generated by the explorer. The evaluator prompt is in the following format:

Tracker Prompt for Single and Comprehensive question types

{function}

Description
{description}

Parameters
{parameters}

Use Case(s)
{use_case}

which contains the name, the description, the parameters and the use cases of the functions.

F Limitations and Broader Impacts

Although precise question answering datasets AirQA for academic papers can enhance research efficiency, this work still has certain limitations: 1) As large language models incorporate more academic papers during pre-training, some questions can be answered solely based on their parametric knowledge; 2) The current dataset is limited to English-language papers in the field of artificial intelligence, and its coverage remains to be improved; 3) While most questions can be evaluated using objective scoring functions, long-form answers inevitably rely on large model-based evaluation, which may affect the consistency and stability of the evaluation results as the continual training and update of these LLMs. For broader social impact, as LLM-based agents become increasingly robust and practical through more refined agent-level finetuning, their improved question-answering capabilities can help researchers save significant time on literature review and detail retrieval, avoid reinventing the wheel, and even assist in building personalized knowledge bases of academic papers.

G Ethics Statement

To construct our AirQA dataset, we use 13,948 papers in artificial intelligence domain. Most papers utilized in our AirQA dataset can be downloaded from `arXiv` [2]. For some conference papers unavailable on `arXiv`, we use `OpenReview` [3] and `ACL Anthology` [4] as supplements. All three websites are licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0), and we use the papers in accordance to their usage terms, with no private, sensitive, or personally identifiable information used in this work.

[2] `https://info.arxiv.org/help/api/index.html`

[3] `https://docs.openreview.net/reference/api-v2`

[4] `https://aclanthology.org/`