AVATAR: Optimizing LLM Agents for Tool Usage via Contrastive Reasoning

Shirley Wu[§], **Shiyu Zhao**[§], **Qian Huang**[§], **Kexin Huang**[§], **Michihiro Yasunaga**[§], **Kaidi Cao**[§]
Vassilis N. Ioannidis[†], **Karthik Subbian**[†], **Jure Leskovec**[*§], **James Zou**[*§]
[*]Equal senior authorship.
[§]Department of Computer Science, Stanford University [†]Amazon

Abstract

Large language model (LLM) agents have demonstrated impressive capabilities in utilizing external tools and knowledge to boost accuracy and reduce hallucinations. However, developing prompting techniques that enable LLM agents to effectively use these tools and knowledge remains a heuristic and labor-intensive task. Here, we introduce AVATAR, a novel and automated framework that optimizes an LLM agent to effectively leverage provided tools, improving performance on a given task. During optimization, we design a comparator module to iteratively deliver insightful and comprehensive prompts to the LLM agent by contrastively reasoning between positive and negative examples sampled from training data. We demonstrate AVATAR on four complex multimodal retrieval datasets featuring textual, visual, and relational information, and three general question-answering (QA) datasets. We find AVATAR consistently outperforms state-of-the-art approaches across all seven tasks, exhibiting strong generalization ability when applied to novel cases and achieving an average relative improvement of 14% on the Hit@1 metric for the retrieval datasets and 13% for the QA datasets. Code and dataset are available at `https://github.com/zou-group/avatar`.

1 Introduction

Autonomous agents powered by large language models (LLMs) offer substantial promise for complex problem-solving [6, 39, 41, 55, 65]. These agents demonstrate remarkable capabilities in reasoning [46, 47, 54, 55] and planning [8, 13, 14, 62]. Additionally, their functionality is extended through the use of external tools that provide access to external or private data and specialized operations, such as APIs for interacting with knowledge bases and search engines. These tools enable agents to perform complex tasks like multi-step problem-solving and retrieving diverse information, which is essential for complex retrieval and question-answering (QA) [13, 21, 26, 33, 38, 40, 48].

Despite the promising capabilities of LLM agents, it remains challenging to engineer effective prompts that guide these agents through a multi-stage process for real-world problem-solving. This process involves (1) decomposing a complex question into an actionable plan with simpler steps, (2) strategically using provided tools to gather relevant information, and, finally, (3) synthesizing intermediate results to produce a coherent and accurate response. Each step requires extensive manual effort and numerous iterations of trial and error to refine the prompts.

Current approaches have primarily focused on directly deploying agents using complex human-designed "mega-prompts" [18, 24, 55], which require lots of manual trial and error. Nevertheless, such hand-engineered mega-prompts may also result in brittle implementations with suboptimal accuracy (see Figure 2 (a)), where the ReAct agent [55] easily produces trivial and misleading answers to

Correspondence: `{shirwu, jure, jamesz}@cs.stanford.edu`

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

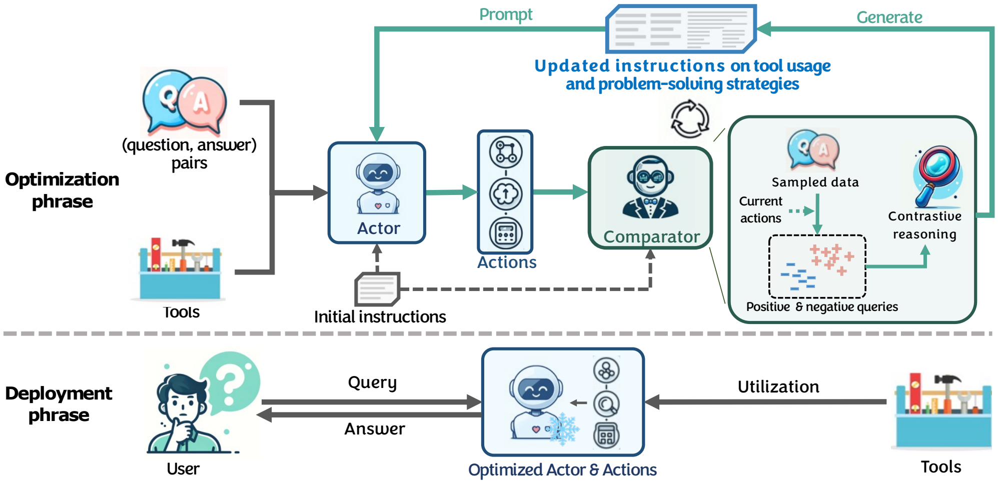


Figure 1: **Overview of AvaTaR.** AvaTaR consists of a actor LLM and a comparator LLM. (a) During optimization, the actor generates actions to answer queries by leveraging the provided tools. Then, the comparator contrasts a set of well-performing (positive) and poorly-performing (negative) queries, automatically generating holistic prompts to teach the actor more effective retrieval strategies and tool usage (*cf.* Section 4). (b) At deployment, the actor with optimized prompts or actions can be effectively used to answer new queries.

customers' queries about specific products. Furthermore, existing research [4, 5, 45, 50, 56, 60, 64] on employing LLMs as optimizers often fails to adequately refine the complex strategies for enhancing tool integration and usage. This lack of strategic optimization can lead to less effective, non-generalizable agent applications in complex real-world scenarios.

Present work: AvaTaR. To address these challenges, we introduce AvaTaR, an automated framework that optimizes agents for effective tool utilization and excellent task performance. Specifically, we leverage key insights from contrastive reasoning and build a comparator module ("trainer") to generate holistic instructions and prompts (*i.e.,* , computing a robust "gradient") to optimize an actor LLM. We demonstrate our framework on challenging tasks of knowledge base retrieval, which involve complex multi-stage procedures and extensive tool usage, and general QA tasks. Specifically, AvaTaR includes two phases:

- **Optimization phase**. The core of our optimization framework (Figure 1) is a comparator LLM that automatically generates holistic prompts to teach a actor LLM to differentiate between effective and ineffective tool usage. The comparator takes positive and negative data samples, where the current agent performs well and poorly, respectively, to identify overall gaps and systematic errors exhibited by the agent. Unlike per-sample instructions, which can easily lead to overfitting on individual data points, by constructing multiple samples as a "batch," the comparator can extract a more robust "gradient" to "backpropagate" to the actor. In other words, the comparator can provide more effective and adaptive prompts through batch-wise contrastive reasoning, helping the agent identify flaws in solving challenging multi-stage problems. Following previous methods [30, 41, 56, 63], we also maintain a memory bank with selected past instructions to prevent the actor LLM from repeating previous mistakes.
- **Deployment phase**. After the optimization phase, the actor with best-performing prompts can be selected for the testing instances. Moreover, in complex retrieval tasks, the iterative optimization through our AvaTaR framework updates the actor for more effective and generalizable action sequences, enabling direct generalization to novel user inquiries at deployment. In Figure 2 (b), the optimized actor creates three novel strategies: 1) precise decomposition of problems by extracting multifaceted attributes, 2) effective tool usage through a sophisticated and robust scoring system, and 3) the strategic combination of different scores, determined by learned coefficients, ensuring accurate and comprehensive retrieval.

Experimental evaluation. We conduct extensive experiments on four retrieval datasets and three QA datasets. The retrieval tasks are highly complex, involving multimodal data, including textual, visual, and relational information. AvaTaR consistently outperforms state-of-the-art methods, showing a substantial 14% improvement in the Hit@1 metric. Impressively, with only 25 iterations, AvaTaR boosts the Hit@1 metric from an initial 5.1% to 28.6% on Flickr30K-Entities [35] and the Recall@20 metric from 30.3% to 39.3% on STaRK-Prime [49]. For general QA datasets, AvaTaR outperforms state-of-the-art methods by 13% on average. These improvements, achieved through iterative updates to the prompts, underscore AvaTaR's ability to optimize agents for complex tasks and effective tool usage. Our key contributions are:

- We introduce AvaTaR, a novel framework that optimizes an actor for effective tool utilization through a comparator module that automatically generates holistic prompts.
- We demonstrate AvaTaR on four complex retrieval tasks and three QA tasks, where it significantly outperforms existing agent methods in terms of task performance and generalization ability.
- We provide a comprehensive analysis of the actor's evolution during optimization, highlighting how comparator automatically provides targeted instructions that improve and generalize the actor.

2 Related Work

LLM Agents. Recent research has leveraged the remarkable language understanding and reasoning abilities of LLMs [1, 41, 47, 54, 55] to complete downstream tasks. For complex tasks that require enhanced capabilities, previous works have positioned LLMs as agents that can interact with environments [4, 6, 13, 18, 21, 26, 27, 40, 48, 55], leverage external tools [6, 28, 31, 33, 36, 38, 39, 66, 68], and gather experiences [7, 61]. For example, ReAct [55] conducts reasoning and action in an interleaved way, retrieving information from Wikipedia to support reasoning.

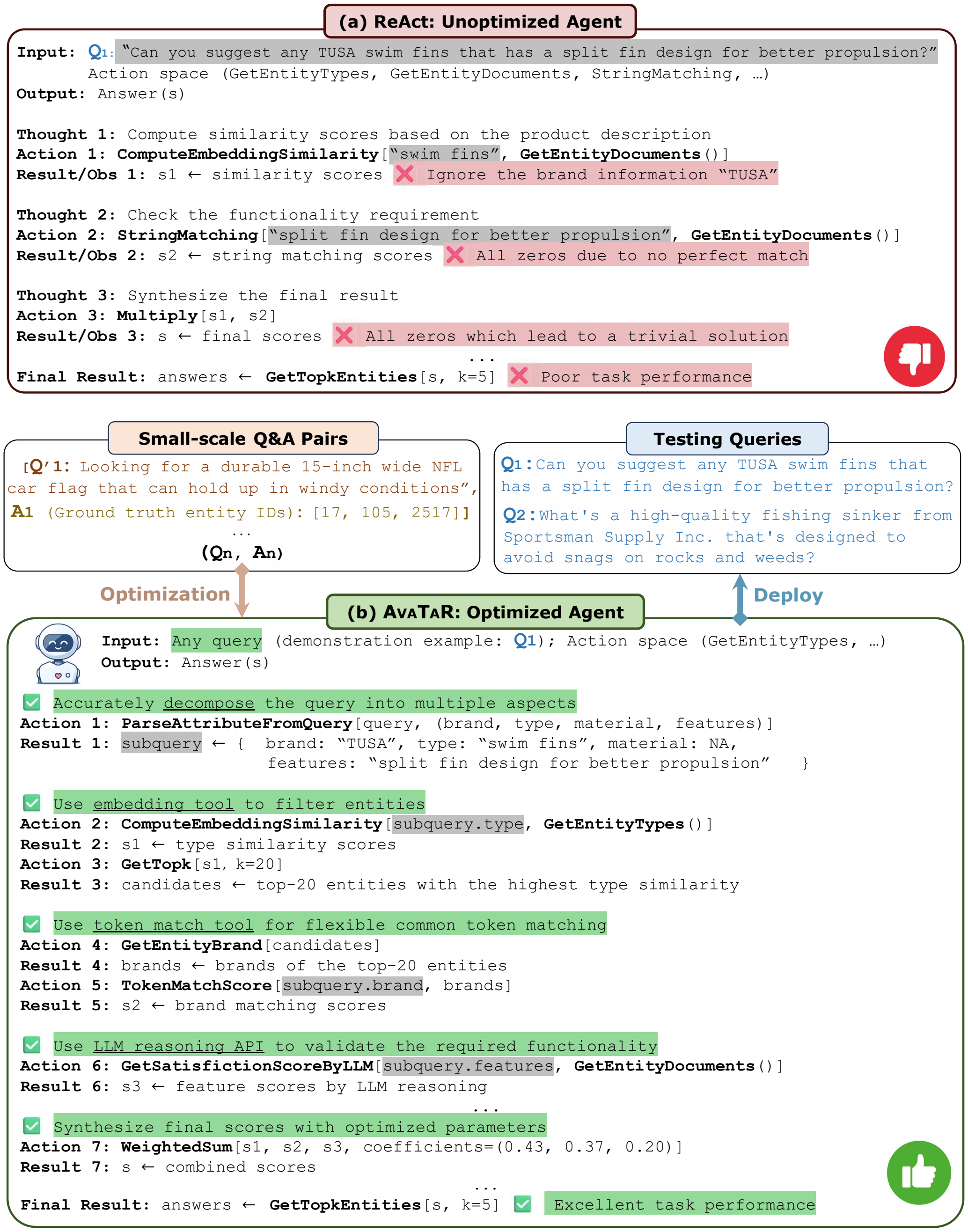
LLM Agents for Retrieval. Previous research has applied LLM agents to Information Retrieval (IR) systems through pretraining [2, 9, 16, 57], reranking [12, 42], and prompting techniques [11, 18]. In IR systems, the retriever module directly influences the performance of downstream tasks, such as retrieval-augmented generation [20, 29, 30] and knowledge-intensive question answering [34, 52]. For example, EHRAgent [40] is designed for EHR question-answering, capable of retrieving relevant clinical knowledge through a structured tool-use planning process and an interactive coding mechanism. However, these LLM agents usually employ heuristic (zero-shot) prompts or rely on few-shot examples [18, 25, 40, 55] for downstream tasks, which lack more informed guidance on generating effective retrieval strategies and tool-assisted actions.

Agent Optimization. In the field of optimizing LLM agents, previous works have modified the parameters of LLM backbones through fine-tuning or instruction tuning to enhance agent capability [3, 15, 19, 23, 32, 33, 37, 43, 51, 58, 59] or generated better prompts through iterative prompt tuning [11, 18, 45, 50, 56]. Recently, Zhang et al. [60] conducted agent training by iteratively updating the agents' functions according to the execution history. However, these methods do not explicitly consider targeted optimization for tool usage or the impact on complex multi-stage tasks. Additionally, enhancing agents' generalization abilities [10, 31, 44], essential for real-world applications, has received less attention. In our work, we focus on automatically generating holistic instructions via a novel contrastive reasoning mechanism, targeting effective tool usage and agents' generalization ability. Compared to fine-tuning approaches, AvaTaR only require a small subset of training data and tool descriptions, making it more adaptable and less computationally intensive.

3 Problem Formulation

Definition 1: Tools. We define tools or APIs as a set of implemented functions with specified input and output variables. We denote the abstract tool space as $\mathcal{T} = \{f_k : \mathcal{I}_{f_k} \to \mathcal{O}_{f_k} \mid k = 1, 2, \ldots\}$, where f_k maps the input $\mathcal{I}_{f_k}$ to the output $\mathcal{O}_{f_k}$. For example, the tools can be APIs used for accessing external knowledge via a search index, an encoder model that generates vector representations from text or image data, or a task-specific classifier that outputs probabilities over a list of classes.

Definition 2: Agents. An LLM agent, defined as $\mathcal{A} : \mathcal{P} \to \alpha$, is controlled by verbal prompts to generate a flow of actions needed to complete a task. Here α denotes the action sequence $[\alpha_1, \ldots, \alpha_L]$, where each action is defined by a tuple $(f \in \mathcal{T}, i \in \mathcal{I}_f, o \in \mathcal{O}_f)$, consisting of a tool function, specified input(s), and a designated variable that receives the output(s). Each action in the sequence can leverage the outputs generated by previous actions, with the final action α_L rendering the results for the task.



```
Input: Q1: "Can you suggest any TUSA swim fins that has a split fin design for better propulsion?"
       Action space (GetEntityTypes, GetEntityDocuments, StringMatching, …)
Output: Answer(s)

Thought 1: Compute similarity scores based on the product description
Action 1: ComputeEmbeddingSimilarity["swim fins", GetEntityDocuments()]
Result/Obs 1: s1 ← similarity scores ✗ Ignore the brand information "TUSA"

Thought 2: Check the functionality requirement
Action 2: StringMatching["split fin design for better propulsion", GetEntityDocuments()]
Result/Obs 2: s2 ← string matching scores ✗ All zeros due to no perfect match

Thought 3: Synthesize the final result
Action 3: Multiply[s1, s2]
Result/Obs 3: s ← final scores ✗ All zeros which lead to a trivial solution
                                    ...
Final Result: answers ← GetTopkEntities[s, k=5] ✗ Poor task performance
```

[Q'1: Looking for a durable 15-inch wide NFL car flag that can hold up in windy conditions", A1 (Ground truth entity IDs): [17, 105, 2517]]
...
(Qn, An)

Q1: Can you suggest any TUSA swim fins that has a split fin design for better propulsion?

Q2: What's a high-quality fishing sinker from Sportsman Supply Inc. that's designed to avoid snags on rocks and weeds?

```
Input: Any query (demonstration example: Q1); Action space (GetEntityTypes, …)
Output: Answer(s)

✓ Accurately decompose the query into multiple aspects
Action 1: ParseAttributeFromQuery[query, (brand, type, material, features)]
Result 1: subquery ← {  brand: "TUSA", type: "swim fins", material: NA,
                        features: "split fin design for better propulsion"   }

✓ Use embedding tool to filter entities
Action 2: ComputeEmbeddingSimilarity[subquery.type, GetEntityTypes()]
Result 2: s1 ← type similarity scores
Action 3: GetTopk[s1, k=20]
Result 3: candidates ← top-20 entities with the highest type similarity

✓ Use token match tool for flexible common token matching
Action 4: GetEntityBrand[candidates]
Result 4: brands ← brands of the top-20 entities
Action 5: TokenMatchScore[subquery.brand, brands]
Result 5: s2 ← brand matching scores

✓ Use LLM reasoning API to validate the required functionality
Action 6: GetSatisfictionScoreByLLM[subquery.features, GetEntityDocuments()]
Result 6: s3 ← feature scores by LLM reasoning
                                    ...
✓ Synthesize final scores with optimized parameters
Action 7: WeightedSum[s1, s2, s3, coefficients=(0.43, 0.37, 0.20)]
Result 7: s ← combined scores
                                    ...
Final Result: answers ← GetTopkEntities[s, k=5] ✓ Excellent task performance
```

Figure 2: **Comparison between AvaTaR and ReAct**. (a) The ReAct agent exhibits incomplete task decomposition and employs suboptimal tool combinations, such as lengthy string matching, leading to poor task performance. (b) AvaTaR decomposes the task into multiple steps, such as type filtering and flexible token matching. Moreover, it implements robust tool usage and precise synthesis with learned parameters from the optimization phase to achieve excellent performance on new queries.

Multi-step problem-solving. Real-world problems are inherently complex and cannot be effectively addressed through straightforward solutions or simple tool usage alone. Solving real-world problems with LLM agents can be structured into a multi-stage procedure:

- **Decomposition of the problem**: The procedure begins by breaking down a complex question into an actionable plan characterized by simpler steps. This decomposition is crucial for setting clear objectives and facilitating focused problem-solving.
- **Tool-assisted subproblem solving**: In the subsequent phase, agents strategically utilize tools from the established tool space $\mathcal{T}$ to gather solutions for each step. This stage is essential for acquiring

Table 1: **Key differences between AVATAR and prevailing agent methods.** AVATAR demonstrates the ability to: 1) self-improve on specific tasks, 2) retain memory throughout the optimization process, 3) enhance the agent's generalization capability, and 4) autonomously generate holistic, high-quality prompts for better tool usage. Please refer to Section 4 for details.

	Self-Improvement	Memory	Generalization	Holistic Prompt Generation (on Tool Usage)
ReAct [55]	✗	✗	✗	✗
Self-refine [27]	✓	✗	✗	✗
Reflexion [41]	✓	✓	✗	✗
AVATAR (Ours)	✓	✓	✓	✓

the necessary information required to effectively address each subproblem of the decomposed problem.

- **Synthesis and response formulation**: The final stage involves synthesizing the intermediate results to construct a precise response. This synthesis not only combines the data but may also refine the response through trials and adjustments, ensuring the solution's accuracy and relevance.

For example, retrieval tasks are inherently complex and demanding. Given a user query q, retrieval tasks aim to identify or generate a ranked list of relevant entities E from the entity space of a knowledge base. Each query is associated with a set of ground truth answers, denoted as Y, which are used to compute the quality of the prediction. Specifically, the LLM agent is required to 1) comprehend a user's request, 2) utilize the provided tools to identify and analyze relevant information in the large knowledge space, which may contain multimodal data sources, and finally, 3) integrate all gathered information to reason and generate an accurate response.

4 Our Method: Optimizing Agents for Tool-Assisted Multi-Step Tasks

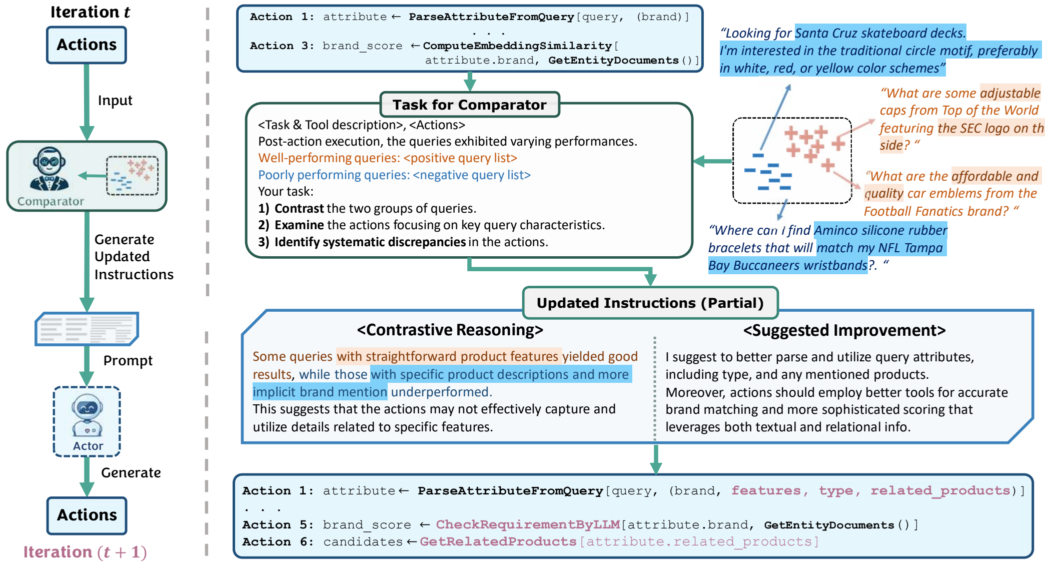
Each step in the multi-stage problem-solving process (described in Section 3) requires effective prompts to identify key flaws and improve task performance. However, refining the agents' prompts demands extensive manual effort and numerous iterations of trial and error.

To address this, we introduce an automated and novel optimization framework, AVATAR, which generates prompts to improve agents' tool usage and task performance. In Table 1, we highlight four critical aspects of our approach compared with prevailing agent frameworks [27, 41, 55]. Here, we introduce the two main LLM components in AVATAR: a actor LLM (Section 4.1) and a comparator LLM (Section 4.2).

4.1 Actor Construction and Challenges

Actor. The actor agent, as defined in Section 3, is responsible for generating initial actions based on the initial instructions/prompts and adjusting actions according to updated instructions. Specifically, the initial instructions provide details about the task and available tools, where tools can be introduced in programming languages such as Python. During optimization, the prompts further incorporate the previous action sequence and updated instructions to adjust these actions. The actor then generates revised actions, which could include a combination of tool usage through programming language (code generation) along with natural language explanations of how the tools are employed.

Challenges in multi-step complex tasks. A common approach to updating instructions utilizes execution results or performance data from a specific instance, often through techniques like self-explanation [4, 27] or self-reflection [41, 56]. However, this approach may not be suitable for complex tasks involving tool usage. Complex multi-step tasks include multiple interacting factors that influence overall performance, such as problem decomposition and tool selection. Consequently, instructions generated for a failed/negative query instance tend to be narrow in scope and may fail to identify flaws across all components of a complex solution. Additionally, while certain tool combinations may be effective for one type of input, their effectiveness can vary across different scenarios, potentially leading to decreased performance when applied to varied cases.



```
Action 1: attribute ← ParseAttributeFromQuery[query,  (brand)]
                                  . . .
Action 3: brand_score ←ComputeEmbeddingSimilarity[
                         attribute.brand, GetEntityDocuments()]
```

<Task & Tool description>, <Actions>
Post-action execution, the queries exhibited varying performances.
Well-performing queries: <positive query list>
Poorly performing queries: <negative query list>
Your task:
1) **Contrast** the two groups of queries.
2) **Examine** the actions focusing on key query characteristics.
3) **Identify systematic discrepancies** in the actions.

“Looking for Santa Cruz skateboard decks. I'm interested in the traditional circle motif, preferably in white, red, or yellow color schemes”

“What are some adjustable caps from Top of the World featuring the SEC logo on the side? “

“What are the affordable and quality car emblems from the Football Fanatics brand? “

“Where can I find Aminco silicone rubber bracelets that will match my NFL Tampa Bay Buccaneers wristbands?. “

<Contrastive Reasoning>

Some queries with straightforward product features yielded good results, while those with specific product descriptions and more implicit brand mention underperformed.
This suggests that the actions may not effectively capture and utilize details related to specific features.

<Suggested Improvement>

I suggest to better parse and utilize query attributes, including type, and any mentioned products.
Moreover, actions should employ better tools for accurate brand matching and more sophisticated scoring that leverages both textual and relational info.

```
Action 1: attribute← ParseAttributeFromQuery[query, (brand, features, type, related_products)]
. . .
Action 5: brand_score ← CheckRequirementByLLM[attribute.brand, GetEntityDocuments()]
Action 6: candidates←GetRelatedProducts[attribute.related_products]
```

Figure 3: **Demonstration example during optimization.** Best viewed in color. The task of the comparator is to automatically generate instructions based on sampled positive and negative queries. Then comparator provides holistic instructions that guide the actor to improve query decomposition, utilize better tools, and incorporate more comprehensive information.

4.2 Automate Holistic Instruction Generation with Comparator

To address these challenges, we construct a comparator LLM to update the instructions for the actor. Instead of optimizing on a sampled instance, comparator aims to identify systematic flaws throughout the structured actions/solutions.

Step 1: Constructing positive and negative queries. To achieve this goal, as shown in Figure 1, the comparator samples a set of data (question-answer pairs), evaluates the current action sequence on the queries, and categorizes them into well-performing (positive) and poorly-performing (negative) groups based on their performance. Specifically, we define two thresholds, ℓ and h (where $0 < h \leq \ell < 1$), which serve as the upper and lower bounds for constructing positive and negative queries, respectively. Queries with an evaluation metric (e.g., Recall) value above ℓ are classified as positive, while those below h are classified as negative. Based on the training dynamics, one could consider adapting the lower bound to ensure a sufficient number of negative samples for selection. After classification, we use random sampling to create a mini-batch of b queries, with an equal split of positive and negative queries ($b/2$ each) for contrastive reasoning.

Step 2: Generating instructions through contrastive reasoning. After this, the comparator is tasked with contrasting the two groups of queries based on their key characteristics, attributing the performance gap to specific tool usage within the complex solution, and finally suggesting general modifications that can improve overall task performance. The instructions generated by the comparator are then appended to the initial prompts to update the actor.

Insights/Justification for the comparator. To illustrate the insights, we draw an analogy from deep neural network training, where extremely small batch sizes can introduce significant noise in gradient estimates and high variance in model updates. By adopting a batched training strategy and sampling positive and negative queries as two “mini-batches,” comparator can extract a robust “gradient” to update the actor. This approach encourages comparator to generate more general and comprehensive instructions on the complex action sequence, including problem decomposition, solutions to subproblems, and the final synthesis. Moreover, as contrastive reasoning directly targets disentangling the performance gap related to input patterns and how they are handled differently by the tools, it is particularly effective in helping comparator differentiate and select tools for use. Finally, by identifying systemic flaws across a wide array of negative queries, comparator generates modifications that are not only tailored to individual samples but also to diverse data samples, enhancing generalization to novel cases.

Demonstration example. Figure 3 illustrates an example where comparator contrasts the patterns of positive and negative queries, identifying discrepancies in tool usage within the action sequence. It reveals that, compared to positive queries, negative queries feature more complex product descriptions, more subtle brand mentions, and additional relevant product mentions. These observations suggest: 1) an incomplete problem decomposition involving query attributes like detailed product features, 2) a potentially imprecise brand match using embedding similarity, and 3) a lack of consideration for related products in the results. Informed by these insights, actor updates its action sequence to address these subproblems and use the tools more effectively for the task, such as replacing the embedding tool with an LLM verification tool.

4.3 Logistic Instructions and Memory Construction

Logistic instructions. While instructions from the comparator are designed to improve task performance, we incorporate two types of orthogonal instructions to ensure the actions are valid and can be executed efficiently.

- **Validity check**: This instruction is triggered internally during the execution of each action. It ensures the validity of the actor's actions, such as verifying the correct use of function calls.
- **Timeout error**: To prevent inefficient action sequences that may stall the actor, we implement a timeout mechanism that triggers an error if processing exceeds a specified threshold. This error prompts the actor to adopt more efficient strategies, such as eliminating redundant operations.

Memory Bank. During optimization, we utilize a memory bank inspired by human decision-making processes, following Shinn et al. [41], where humans typically address current problems by analyzing the current situation and referencing past experiences. The memory bank stores tuples of action sequences, instructions from comparator, and the performance of these action sequences on a small training set (sampled from positive and negative queries). To manage the context size input to actor, we retain only the top-5 action sequences with the best performance. This memory bank enables actor to learn from both immediate instructions and historical results.

Deployment. At deployment, we can apply the optimized instructions or, as shown in Figure 1, the optimized actor /action sequence, which includes effective tool utilization and problem-solving strategies, to answer queries or retrieve entities. In the experiments, we demonstrate AVATAR's flexibility by applying different deployment strategies.

5 Experiments

Tasks and Evaluation. We conduct experiments on the following datasets:

- **Four challenging retrieval datasets from STARK [49] and FLICKR30K-ENTITIES [35]** to demonstrate AVATAR in handling complex real-world tasks (*cf.* details in Appendix A). For each query in the retrieval datasets, the task is to retrieve relevant entities, such as nodes in a knowledge graph or images in knowledge bases. During deployment, we directly apply the optimized action sequence to the test queries. We assess task performance by comparing the consistency of the results with the ground truth answers in the datasets, using Hit@1, Hit@5, Recall@20, and Mean Reciprocal Rank (MRR) as the metrics.
- **Three question-answering (QA) benchmarks: HotpotQA [53], ArxivQA [22], ToolQA [67]**, where the task is to provide natural language answers to the questions. We sample 100, 100, and 40 training queries, and 100, 100, and 60 testing queries for the three benchmarks, respectively. During deployment, the actor LLM uses optimized instructions to generate the action sequence for obtaining the answer. We use exact match (EM) score on HotpotQA, following previous methods. For ArxivQA and ToolQA, we use the LLM judge score for more reliable evaluation.

Baselines. For the knowledge retrieval tasks, we employ several embedding-based retriever models for our evaluation, following Wu et al. [49]: Dense Passage Retriever (DPR) Karpukhin et al. [17]; Vector Similarity Search methods ada-002 and multi-ada-002 using `text-embedding-ada-002` from OpenAI; and a relation-aware model, QAGNN [57], for the STARK benchmark. Additionally, we include four prevailing agent frameworks to further enrich our evaluation:

- **ReAct** [55] conducts reasoning and action in an in-context and interleaved manner to enable LLMs to interactively analyze observed information and perform actions.

Table 2: Retrieval performance (%) on STaRK benchmark. Last row shows the relative improvements over the best metric value in each column.

	AMAZON				MAG				PRIME			
	Hit@1	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**
DPR	15.29	47.93	44.49	30.20	10.51	35.23	42.11	21.34	4.46	21.85	30.13	12.38
QAGNN	26.56	50.01	52.05	37.75	12.88	39.01	46.97	29.12	8.85	21.35	29.63	14.73
ada-002	39.16	62.73	53.29	50.35	29.08	49.61	48.36	38.62	12.63	31.49	36.00	21.41
multi-ada-002	40.07	64.98	55.12	51.55	25.92	50.43	**50.80**	36.94	15.10	33.56	38.05	23.49
ReAct	42.14	64.56	50.81	52.30	31.07	49.49	47.03	39.25	15.28	31.95	33.63	22.76
Reflexion	42.79	65.05	54.70	52.91	40.71	54.44	49.55	47.06	14.28	34.99	38.52	24.82
AVATAR-C	40.92	63.63	53.68	51.73	33.25	52.17	47.88	41.34	8.82	23.82	30.32	16.20
AVATAR	**49.87**	**69.16**	**60.57**	**58.70**	**44.36**	**59.66**	50.63	**51.15**	**18.44**	**36.73**	**39.31**	**26.73**
Relative Improvement	16.6%	6.3%	9.9%	12.2%	9.6%	2.1%	-0.3%	8.7%	20.7%	5.0%	2.1%	7.7%

- **Reflexion** [41] uses self-reflection on the current task completion and stores these reflections in an episodic memory buffer to enhance decision-making in subsequent trials.
- **ExpeL** [61] extracts insights from successful and failed action sequences, retrieving and including them in the context during inference. We apply ExpeL on the QA datasets and, due to its high cost on large-scale retrieval tasks, compare it with AVATAR on a sampled STARK-MAG test set.
- **Retroformer** [56] reinforces LLM agents and automatically tunes their prompts by learning a retrospective model through policy gradient. We compare the performance of AVATAR with the reported result by Retroformer on HotpotQA due to the additional training involved.

We include an ablation model, AVATAR-C, which removes the comparator from our optimization pipeline. This comparison aims to validate the effectiveness of the comparator. The LLM version information is provided in Appendix B.

Function library. For the knowledge retrieval tasks, our function library consists of twenty-eight functions that facilitate access to, operation on, and reasoning over the knowledge information by LLM agents. For the QA tasks, we provide web search tools such as Google and Arxiv search APIs. See Appendix E for details. We used the same function library across all agent methods.

General pipeline. For AVATAR, we optimize the agent for a fixed number of epochs and select the action sequence or instruction with the highest performance. We use the same initial prompt structure, the metric Recall@20 or Accuracy for constructing positive and negative queries, and hyperparameters ($\ell = h = 0.5$, $b = 20$) for all datasets.

5.1 Textual and Relational Retrieval Tasks

We employ the AMAZON, MAG, and PRIME datasets from the STARK benchmark [49], a large-scale semi-structured retrieval benchmark that integrates textual and relational knowledge (*cf.* detailed description in Appendix A). Here, the entities to be retrieved are defined as nodes in a graph structure, with knowledge associated with each entity including both textual descriptions and relational data. We use the official splits from the STARK benchmark.

Takeaway 1: AVATAR outperforms state-of-the-art models. Table 3 shows that AVATAR substantially outperforms leading models such as Reflexion across all metrics on the STARK benchmark. Notably, the average improvement of AVATAR is 15.6% on Hit@1 and 9.5% on MRR. ReAct agents, however, cannot optimize based on instructions for improved tool usage and tend to select tools based on the LLM's prior knowledge, which may not be optimal for the given task. We observe that ReAct agents apply similar tools across various queries and struggle to explore alternative tool usage even with extensive in-context reasoning. Results for agent methods using GPT-4 Turbo are provided in Appendix B, showing similar conclusions. For comparison with ExpeL, the results in Table 6 show that it performs similarly to ReAct, underperforming AVATARby a large margin.

Takeaway 2: Comparator greatly impacts the actor's performance. The comparison of AVATAR with its ablation variant, AVATAR-C, highlights the significant advantages of the comparator module. Although AVATAR-C conducts validity and timeout checks, integrating Comparator into AVATAR adds a comprehensive instruction mechanism crucial for identifying clear directions to improve the agents, underlining comparator's key role in optimizing actor.

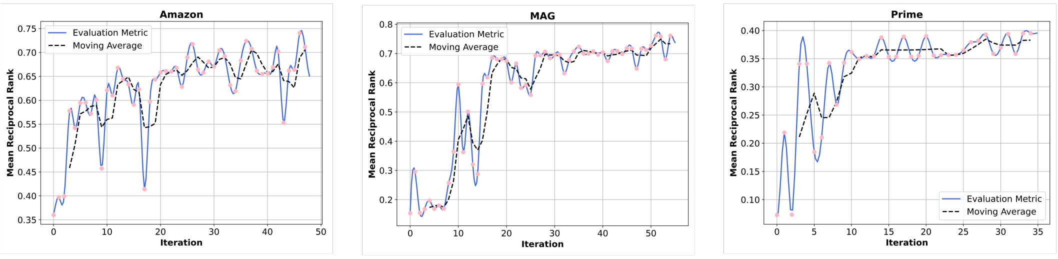


Figure 4: **Optimization dynamics of AvaTaR agents on STaRK**. The figures show validation performance (solid line) and its moving average (dashed line) during the optimization of AvaTaR.

	Hit@1	Hit@5	R@20	MRR
`clip-vit-large-patch14`	37.2	56.4	72.8	46.3
ReAct (`claude3`)	38.8	54.8	71.6	46.1
Reflexion (`claude3`)	28.4	53.2	75.2	41.2
AvaTaR-C (`claude3`)	28.8	53.2	78.4	40.0
AvaTaR (`claude3`)	**42.4**	**63.0**	**79.2**	**52.3**
Relative Improvement	9.2%	11.7%	5.3%	13.0%

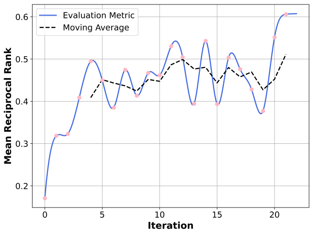


Figure 5: Performance (left) and AvaTaR's optimization dynamics (right) on Flickr30K-Entities.

Takeaway 3: AvaTaR effectively improves agents during optimization. Figure 4 illustrates the agents' performance on the validation set during optimization. Impressively, AvaTaR agents show significant performance improvements, e.g., from 35% to 75% on Amazon and from 20% to 78% on MAG. This evidence strongly supports the effectiveness of the instructions generated by our comparator. Additionally, our memory bank, which stores past best-performing actions, encourages AvaTaR agents to gradually converge by the end of the optimization process.

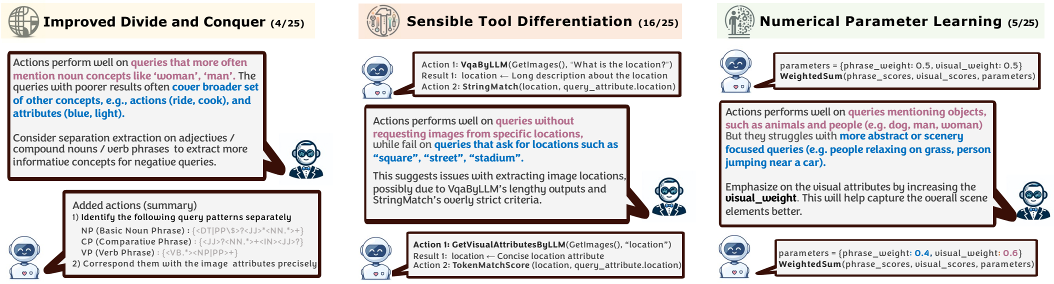
Takeaway 4: AvaTaR can generalize to real-world tasks. Comparator generates instructions tailored to groups of retrieval queries, promoting generalizable modifications for novel queries. We validate this capability by applying optimized actions to human-generated leave-out queries from the STaRK benchmark, which differ notably from the training data used to optimize our agents. Results in Table 5 (Appendix B) show that AvaTaR significantly outperforms other models, achieving an average improvement of 20.9% on Hit@1. Further, in another study of Appendix B, we assess AvaTaR's robustness to hyperparameters h and ℓ, showing that it maintains stable performance and generalization across different parameter values.

5.2 Image Retrieval Task

We further experiment on Flickr30K Entities [35], an image retrieval dataset of 30k images with annotated bounding boxes and descriptive phrases (Appendix A). In Table 2, AvaTaR again shows significant improvements. In contrast, Reflexion agents struggle with "overfitting," where they are easily misled by specific image data, leading to inappropriate actions (e.g., trying to "extract the color of a hat" from images without hats). AvaTaR effectively avoids such pitfalls through batch-wise contrastive reasoning, which provides a broader perspective.

Takeaway 5: AvaTaR generates impressive and generalizable actions. In Figure 5 (left), the final actions of the AvaTaR agent detailed in Figure 8 (Appendix B), achieve advanced performance. Notably, AvaTaR skillfully manages input queries and leverages Inverse Document Frequency (IDF) scores to refine phrase matching, ultimately synthesizing accurate answers. Beyond using existing tools, AvaTaR agents can develop high-level tools, such as IDF-based reweighting, suggesting a promising direction for dynamic tool libraries and enhanced tool generation.

Takeaway 6: Emerging behaviors during optimization. In Figure 6, we present concrete cases illustrating key interactions between actor and comparator. In each instance, comparator identifies critical flaws, including information omission, ineffective tool usage, and suboptimal synthesis of varying scores. The instructions subsequently prompt actor to enhance retrieval strategies, tool selection, and precise score combinations. Furthermore, frequent references to tool usage underscore comparator's focused examination of tool utilization during optimization.


Improved Divide and Conquer (4/25)
Sensible Tool Differentiation (16/25)
Numerical Parameter Learning (5/25)

Actions perform well on queries that more often mention noun concepts like 'woman', 'man'. The queries with poorer results often cover broader set of other concepts, e.g., actions (ride, cook), and attributes (blue, light).

Consider separation extraction on adjectives / compound nouns / verb phrases to extract more informative concepts for negative queries.

```
Added actions (summary)
1) Identify the following query patterns separately
   NP (Basic Noun Phrase) : {<DT|PP\$>?<JJ>*<NN.*>+}
   CP (Comparative Phrase) : {<JJ>?<NN.*>+<IN><JJ>?}
   VP (Verb Phrase) : {<VB.*><NP|PP>+}
2) Correspond them with the image  attributes precisely
```

```
Action 1: VqaByLLM(GetImages(), "What is the location?")
Result 1:  location ← Long description about the location
Action 2: StringMatch(location, query_attribute.location)
```

Actions performs well on queries without requesting images from specific locations, while fail on queries that ask for locations such as "square", "street", "stadium".

This suggests issues with extracting image locations, possibly due to VqaByLLM's lengthy outputs and StringMatch's overly strict criteria.

```
Action 1: GetVisualAttributesByLLM(GetImages(), "location")
Result 1:  location ← Concise location attribute
Action 2: TokenMatchScore (location, query_attribute.location)
```

```
parameters = {phrase_weight: 0.5, visual_weight: 0.5}
WeightedSum(phrase_scores, visual_scores, parameters)
```

Actions performs well on queries mentioning objects, such as animals and people (e.g. dog, man, woman) But they struggles with more abstract or scenery focused queries (e.g. people relaxing on grass, person jumping near a car).

Emphasize on the visual attributes by increasing the **visual_weight**. This will help capture the overall scene elements better.

```
parameters = {phrase_weight: 0.4, visual_weight: 0.6}
WeightedSum(phrase_scores, visual_scores, parameters)
```

Figure 6: **Representative instruction types from the comparator.** We provide three cases where the comparator guides the actor towards (1) better divide-and-conquer strategies for multi-step problem-solving, (2) more sensible differentiation between good and bad tool usage/combinations, and (3) adjustments in the weights to generate the final answers. We record the number of occurrences X under each instruction type over 25 iterations on FLICKR30K-ENTITIES, indicated by (X/25).

Table 3: Performance (%) on three QA benchmarks. Last row shows the relative improvements over the best metric value in each column.

	HOTPOTQA	ARXIVQA	TOOLQA			
			SCIREX-EASY	SCIREX-HARD	AGENDA-EASY	AGENDA-HARD
CoT	28.0%	58.0%	1.7%	0.0%	0.0%	0.0%
ReAct	40.0%	72.0%	31.7%	17.5%	38.3%	3.33%
Reflexion	46.0%	77.0%	28.3%	13.3%	30.0%	3.33%
ExpeL	39.0%	73.0%	36.7%	14.5%	56.6%	1.67%
Retroformer (#retry=1)	51.0%	-	-	-	-	-
AVATAR-C	41.0%	73.0%	31.7%	13.3%	31.7%	1.67%
AVATAR	**53.0%**	**84.0%**	**37.5%**	**23.3%**	**60.0%**	**4.17%**
Relative Improvement	3.92%	9.09%	2.18%	33.1%	5.82%	25.0%

5.3 Question Answering Tasks

Finally, we applied AVATAR to three widely used QA benchmarks. For ToolQA, we tested AVATAR and the baselines on two different domains: SciREX, which focuses on extracting information from full-length machine learning papers, and Agenda, which involves personal agenda-related questions. Both datasets have easy and hard versions.

Takeaway 7: AVATAR outperforms on QA tasks by offering better context understanding. Table 3 shows that AVATAR consistently outperforms state-of-the-art methods across all three QA datasets, with especially strong results on TOOLQA. In SCIREX-HARD, which focuses on extracting complex information from long scientific papers, AVATAR shows a 33.1% improvement, while in AGENDA-HARD, it achieves a 25.0% relative gain. These improvements are attributed to AVATAR's ability to generate optimized prompts that help the agent better understand the broader patterns and contexts of the questions, leading to more accurate answers and improved generalization across question types, from simple to complex.

6 Conclusion and Future Work

In this study, we introduce AVATAR, a novel framework that automates the optimization of LLM agents for enhanced tool utilization in multi-step problems, focusing on complex retrieval and QA tasks. AVATAR demonstrates remarkable improvements across seven diverse datasets. This success can largely be attributed to the comparator module, which effectively refines agent performance through the iterative generation of holistic and strategic prompts. A key innovation of comparator is its use of contrastive reasoning with batch-wise sampling, enabling it to identify systemic flaws and extract robust "gradients" for comprehensive agent improvement across diverse scenarios. While we observe substantial progress from AVATAR, we discuss its limitations in Appendix D regarding its scalability *etc.*Future work can explore extending this methodology to other challenging agent tasks, visual reasoning tasks, and more dynamic environments, or designing better memory banks for dynamically storing knowledge and experience from past training.

Acknowledgement

We thank lab members in Zou and Leskovec's labs for discussions and for providing feedback on our manuscript. We also gratefully acknowledge the support of DARPA under Nos. N660011924033 (MCS); NSF under Nos. OAC-1835598 (CINES), CCF-1918940 (Expeditions), DMS-2327709 (IHBEM); Stanford Data Applications Initiative, Wu Tsai Neurosciences Institute, Stanford Institute for Human-Centered AI, Chan Zuckerberg Initiative, Amazon, Genentech, GSK, Hitachi, SAP, and UCB. The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding entities.

REFERENCES

[1] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. [n. d.]. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. ([n. d.]). arXiv:2308.09687

[2] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving Language Models by Retrieving from Trillions of Tokens. In ICML.

[3] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. FireAct: Toward Language Agent Fine-tuning. arXiv:2310.05915

[4] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching Large Language Models to Self-Debug. (2023).

[5] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. 2022. RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning. In EMNLP. ACL.

[6] Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, Katsushi Ikeuchi, Hoi Vo, Li Fei-Fei, and Jianfeng Gao. [n. d.]. Agent AI: Surveying the Horizons of Multimodal Interaction. ([n. d.]). arXiv:2401.03568

[7] Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. AutoGuide: Automated Generation and Selection of State-Aware Guidelines for Large Language Model Agents. CoRR abs/2403.08978 (2024). arXiv:2403.08978

[8] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. [n. d.]. MindAgent: Emergent Gaming Interaction. ([n. d.]). arXiv:2309.09971

[9] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In ICML. PMLR.

[10] Simon Jerome Han, Keith J. Ransom, Andrew Perfors, and Charles Kemp. 2024. Inductive reasoning in humans and large language models. Cogn. Syst. Res. (2024).

[11] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. (2024). arXiv:2402.07630

[12] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2023. Large language models are zero-shot rankers for recommender systems. arXiv:2305.08845 (2023).

[13] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In ICML.

[14] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models. In CoRL.

[15] Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. MetaTool Benchmark for Large Language Models: Deciding Whether to Use Tools and Which to Use. ICLR (2024).

[16] Vassilis N Ioannidis, Xiang Song, Da Zheng, Houyu Zhang, Jun Ma, Yi Xu, Belinda Zeng, Trishul Chilimbi, and George Karypis. 2022. Efficient and effective training of language and graph neural network models. arXiv preprint arXiv:2206.10781 (2022).

[17] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In EMNLP.

[18] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2023. Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP. arXiv:2212.14024 [cs.CL]

[19] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu-Hong Hoi. 2022. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. In NeurIPS.

[20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL]

[21] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Scale Language Model Society. abs/2303.17760 (2023). arXiv:2303.17760

[22] Lei Li, Yuqi Wang, Runxin Xu, Peiyi Wang, Xiachong Feng, Lingpeng Kong, and Qi Liu. 2024. Multimodal ArXiv: A Dataset for Improving Scientific Comprehension of Large Vision-Language Models. In ACL.

[23] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In EMNLP. Association for Computational Linguistics.

[24] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. AgentBench: Evaluating LLMs as Agents. (2023). arXiv:2308.03688

[25] Robert Lo, Abishek Sridhar, Frank Xu, Hao Zhu, and Shuyan Zhou. 2023. Hierarchical Prompting Assists Large Language Model on Web Navigation. In Findings of the Association for Computational Linguistics: EMNLP 2023.

[26] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models. In NeurIPS.

[27] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In NeurIPS.

[28] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Ouyang Long, 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. 2021. WebGPT: Browser-assisted question-answering with human feedback. ArXiv (2021).

[29] 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. [n. d.]. WebGPT: Browser-assisted question-answering with human feedback. ([n. d.]). arXiv:2112.09332

[30] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. 2310.08560 (2023).

[31] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. ART: Automatic multi-step reasoning and tool-use for large language models. arXiv:2303.09014 (2023).

[32] Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. TALM: Tool Augmented Language Models. arXiv:2205.12255

[33] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large Language Model Connected with Massive APIs. CoRR (2023). arXiv:2305.15334

[34] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. 2023. Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback. arxiv 2302.12813 (2023).

[35] Bryan A. Plummer, Liwei Wang, Chris M. Cervantes, Juan C. Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. 2017. Flickr30k Entities: Collecting Region-to-Phrase Correspondences for Richer Image-to-Sentence Models. Int. J. Comput. Vis. (2017).

[36] Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, Ruobing Xie, Fanchao Qi, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023. WebCPM: Interactive Web Search for Chinese Long-form Question Answering. In Proceedings of ACL 2023. Association for Computational Linguistics.

[37] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. arxiv 2307.16789 (2023).

[38] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In NeurIPS.

[39] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. In NeurIPS.

[40] Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce Ho, Carl Yang, and May D Wang. 2024. EHRAgent: Code Empowers Large Language Models for Complex Tabular Reasoning on Electronic Health Records. arXiv:2401.07128 (2024).

[41] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In NeurIPS.

[42] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. [n. d.]. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. In EMNLP, year = 2023.

[43] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases. (2023). arXiv:2306.05301

[44] Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D. Goodman. 2024. Hypothesis Search: Inductive Reasoning with Language Models. ICLR (2024).

[45] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. 2023. PromptAgent: Strategic Planning with Language Models Enables Expert-level Prompt Optimization. (2023). arXiv:2310.16427

[46] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In ICLR.

[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In NeurIPS.

[48] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. (2023). arXiv:2308.08155

[49] Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. STaRK: Benchmarking LLM Retrieval on Textual and Relational Knowledge Bases. (2024). arXiv:2404.13207

[50] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large Language Models as Optimizers. (2024).

[51] Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. GPT4Tools: Teaching Large Language Model to Use Tools via Self-instruction. In NeurIPS, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.).

[52] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. EMNLP (2018).

[53] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In EMNLP.

[54] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In NeurIPS.

[55] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In ICLR.

[56] Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 2024. Retroformer: Retrospective Large Language Agents with Policy Gradient Optimization. (2024).

[57] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering.

[58] Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. ToolEyes: Fine-Grained Evaluation for Tool Learning Capabilities of Large Language Models in Real-world Scenarios. arXiv:2401.00741 (2024).

[59] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. [n. d.]. AgentTuning: Enabling Generalized Agent Abilities for LLMs. ([n. d.]). arXiv:2310.12823

[60] Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. 2024. Training Language Model Agents without Modifying Language Models. (2024). arXiv:2402.11359

[61] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM Agents Are Experiential Learners. In AAAI.

[62] Wenqing Zheng, SP Sharan, Ajay Kumar Jaiswal, Kevin Wang, Yihan Xi, Dejia Xu, and Zhangyang Wang. 2023. Outline, then details: Syntactically guided coarse-to-fine code generation. ICML (2023).

[63] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing Large Language Models with Long-Term Memory. In AAAI.

[64] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models are Human-Level Prompt Engineers. In ICLR.

[65] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. arXiv:2308.07107 (2023).

[66] Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. ToolQA: A Dataset for LLM Question Answering with External Tools. In NeurIPS.

[67] Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. ToolQA: A Dataset for LLM Question Answering with External Tools. arXiv:2306.13304

[68] Chang Zong, Yuchen Yan, Weiming Lu, Eliot Huang, Jian Shao, and Yueting Zhuang. [n. d.]. Triad: A Framework Leveraging a Multi-Role LLM-based Agent to Solve Knowledge Base Question Answering. ([n. d.]). arXiv:2402.14320

A Retrieval Tasks

STaRK. On the STaRK benchmark, we are given a relation-text knowledge base, based on a knowledge graph $G = (V, E)$ and a collection of free-text documents D. We represent the relation-text knowledge base of size n as $\mathcal{E} = \{(v_i, d_i, g_i)\}_{i=1}^{n}$, where $v_i \in V$ represents a node on the knowledge graph, $d_i \in D$ is the text document related to the node, and g_i is the connected component of G containing v_i.

The query set Q in STaRK is derived from both G and D, where each $q_i \in Q$ contains requirements based on d_i and g_i. The answer set A_i, which includes v_i, is a set of nodes satisfying both relational and textual requirements. The task on STaRK is defined as follows: Given the knowledge base $\mathcal{E}$ consisting of relational and textual information, and a text query q_i, the output is a set of nodes A_i such that $\forall a_i \in A_i$, a_i satisfies the relational requirements in the knowledge graph and textual requirements in its text documents.

Flickr30K Entities. On the Flickr30K Entities dataset, we are given an image-text knowledge base. We denote an image-text knowledge base of size n as $\mathcal{E} = \{(v_i, q_i, T_i)\}_{i=1}^{n}$. Sample i consists of an image v_i, its descriptive caption q_i, and entity bounding box information T_i. Specifically, $T_i = \{(c_{ij}, p_{ij})\}_{j=1}^{b_i}$, where b_i represents the number of bounding boxes annotated in image i, c_{ij} is the coordinate of the j-th bounding box, and p_{ij} describes the entity in the corresponding bounding box.

In our task, the image captions serve as the text query; therefore, all q_i in the dataset are not accessible to the agent to prevent information leakage. However, the agent can access v_i and T_i to fully utilize the vision and language information. The task on Flickr30K Entities is defined as follows: Given the knowledge base $\mathcal{E}$ with images and bounding box information, and a text query q_i, the output is an image v_i that satisfies the visual requirements in the image and textual requirements in the corresponding bounding boxes T_i.

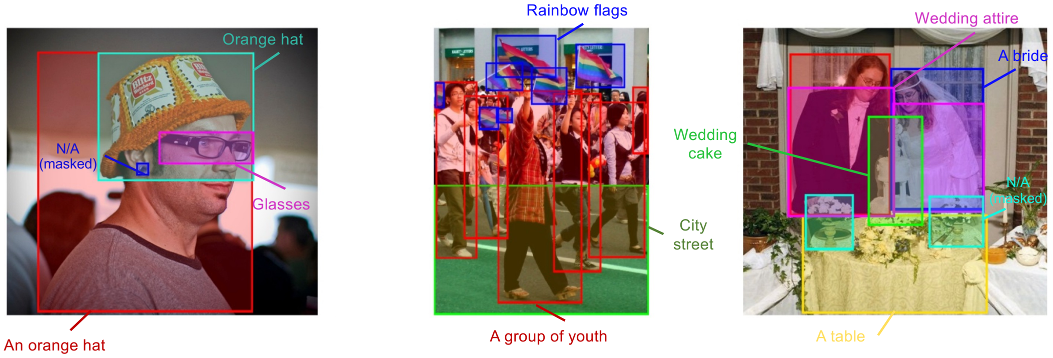


Figure 7: **Example data on Flickr30K Entities.** Each entity is an image along with its image patches and associated phrases with the image patches.

B Experiment Details and Additional Results

B.1 Experiment Setup

LLM versions for agent methods.

- For the knowledge retrieval tasks, we use `claude-3-opus` as the backbone LLM in the main paper by default, and report results using `gpt-4-turbo` in Appendix B due to space limitations.
- For the QA tasks, we use `gpt-4` for HotpotQA for fair comparison with previous methods and `gpt-4o` for the other two QA datasets.

Table 4: Retrieval performance (%) on STaRK benchmark. Last row shows the relative improvements over the best metric value among the baselines.

	Amazon				MAG				Prime			
	Hit@1	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**
DPR (roberta)	15.29	47.93	44.49	30.20	10.51	35.23	42.11	21.34	4.46	21.85	30.13	12.38
QAGNN (roberta)	26.56	50.01	52.05	37.75	12.88	39.01	46.97	29.12	8.85	21.35	29.63	14.73
ada-002	39.16	62.73	53.29	50.35	29.08	49.61	48.36	38.62	12.63	31.49	36.00	21.41
multi-ada-002	40.07	64.98	55.12	51.55	25.92	50.43	**50.80**	36.94	15.10	33.56	38.05	23.49
ReAct (gpt4)	38.83	62.50	50.39	49.16	23.50	46.50	43.11	33.91	10.83	30.83	32.16	19.39
Reflexion (gpt4)	41.45	64.83	53.98	52.22	33.44	51.33	49.14	41.34	14.27	35.11	39.29	23.61
Reranker (gpt4)	44.79	71.17	55.35	55.69	40.90	58.18	48.60	49.00	18.28	37.28	34.05	26.55
AvaTaR-C (gpt4)	32.03	58.46	54.03	44.00	25.97	45.62	46.68	35.12	9.52	26.04	32.62	17.58
AvaTaR (gpt4)	**48.82**	**72.03**	56.04	57.17	**46.08**	59.32	49.70	**52.01**	**20.10**	**39.89**	**42.23**	**29.18**
Relative Improvement (over Best Baseline)	9.0%	1.2%	1.3%	2.7%	12.7%	2.1%	-2.2%	6.1%	10.0%	7.0%	11.0%	9.9%

B.2 Additional Experimental Results

(1) AvaTaR results on STaRK using GPT-4 Turbo (0125) as LLM backbone. In Table 4, we provide the results on STaRK using GPT-4 Turbo (0125) as the backbone LLM.

Table 5: Retrieval performance (%) on the leave-out sets of human-generated queries in STaRK.

	Amazon				MAG				Prime			
	Hit@1	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**	**Hit@1**	**Hit@5**	**R@20**	**MRR**
DPR (roberta)	16.05	39.51	15.23	27.21	4.72	9.52	25.00	7.90	2.04	9.18	10.69	7.05
ada-002	39.50	64.19	35.46	52.65	28.57	41.67	35.95	35.81	17.35	34.69	41.09	26.35
multi-ada-002	46.91	72.84	40.22	58.74	23.81	41.67	**39.85**	31.43	24.49	39.80	47.21	32.98
ReAct	45.65	71.73	35.95	58.81	27.27	40.00	35.95	33.94	21.73	33.33	41.09	28.20
Reflexion	49.38	64.19	35.95	58.96	28.57	39.29	35.95	36.53	16.52	33.03	41.09	23.99
AvaTaR	**58.32**	**76.54**	**42.43**	**65.91**	**33.33**	**42.86**	35.94	**38.62**	**33.03**	**51.37**	**53.34**	**41.00**
Rel. Impr.	17.5%	5.1%		11.8%	16.7%	2.9%		5.7%	28.7%	27.3%		21.4%

(2) AvaTaR results on STaRK's human-generated splits. In Table 5, we demonstrate AvaTaR's ability to generalize to test queries with distributions different from the question-answering pairs used to optimize the actor agents.

Table 6: Performance metrics for different models on the subset of the STaRK-MAG dataset.

	MAG (#Test=50)			
	Hit@1	**Hit@5**	**Recall@20**	**MRR**
DPR	16.00	40.00	51.84	27.39
QAGNN	20.00	52.00	49.71	36.39
ada-002	40.00	58.00	55.93	47.76
multi-ada-002	32.00	58.00	58.81	43.58
ReAct	46.00	60.00	54.67	50.92
ExpeL	40.00	58.00	55.94	47.43
Reflexion	48.00	64.00	57.43	52.31
AvaTaR-C	44.00	60.00	52.49	50.16
AvaTaR	52.00	64.00	53.86	56.74
Relative Improvement	8.33%	0.00%	-8.42%	8.48%

(3) AvaTaR results and comparison with ExpeL on STaRK-MAG subset. In Table 6, AvaTaR demonstrates consistently higher performance than ExpeL across most metrics, notably achieving the highest Hit@1 and MRR scores. While ExpeL performs well in Recall@20, AvaTaR 's overall improvements highlight its superior capability in precise retrieval tasks and tool-assisted knowledge retrieval.

(4) Final action sequence by AvaTaR on Flickr30K-Entities. In Figure 8, we present the final actions optimized by AvaTaR on Flickr30K-Entities.

(5) Sensitivity of AvaTaR to upper and lower bounds. We evaluated various combinations of ℓ and h, focusing on the STaRK-Amazon dataset due to computational constraints. Table 7 presents the Hit@1 results for different ℓ and h values.

```
Input: Any query (example: "A man with pierced ears is wearing glasses and an orange hat .");
       Action Space: {GetImages, GetEmbeddingSimilarity, GetVisualAttributesByLLM, , ...]
Output: Retrieved Image IDs

✅ Remove empty spaces or non-alphabetic characters
Action 1: CleanQueryText[query]
Result 1: normalized_query ← "a man with pierced ears is wearing glasses and an orange hat"

✅ Get all phrases from the knowledge base
Action 2: GetBagofPhrases()
Result 2: phrases_list ← [["a man", "grass", "sky"], ["a", "cat", ...]]

✅ Compute IDF for phrase importance
Action 3: ComputeIDFScores[Flatten[phrases_list]]
Result 3: idf_scores ← {"pierced": 0.5, "man": 0.0012, ...}

✅ Get visual attributes for the candidate images
Action 4: GetVisualAttributesByLLM[GetImages(), ["color", "object", "action", "count"]]
Result 4: visual_attributes ← {node_id_1: {"color": "red", ...}, node_id_2: {...}, ...}

✅ Evaluate textual and visual relevance
Action 5: [‘,‘.join(list) for list in phrases_list]
Result 5: phrase_sentences ← ["a man, grass, sky", "a, cat, playground”,...]

✅ Evaluate textual and visual relevance
Action 6: ComputeEmbeddingSimilarity[normalized_query, phrase_sentences]
Result 6: text_scores

Action 7: ComputeEmbeddingSimilarity[normalized_query, visual_attributes]
Result 7: visual_scores

✅ Match query phrases with node attributes using IDF scores
Action 8: MatchQueryPhrases[normalized_query.split(), visual_attributes]
Result 8: phrase_match_scores

✅ Reweight the phrase_match_scores with IDF score
Action 9: ReweightByIDFScore[phrase_match_scores, idf_scores]
Result 9: reweighted_match_scores

✅ Aggregate scores with weighted parameters
Action 10: WeightedSum[text_scores, visual_scores, reweighted_match_scores, weights=(0.5, 0.3, 0.2)]
Result 10: aggregated_scores

✅ Normalize scores for final ranking
Action 11: NormalizeScores[aggregated_scores]
Result 11: normalized_scores = {node_id: normalized_score, ...}

Final Result: answers = GetTopkEntities[normalized_scores, k=5]
✅ Excellent task performance
```

Figure 8: **Optimized Action Sequence by AVATAR on FLICKR30K-ENTITIES.**.

Table 7: Hit@1 results for different combinations of ℓ and h values on the STARK-AMAZON dataset.

	$h = 0.3$	$h = 0.4$	$h = 0.5$
$\ell = 0.5$	48.32	50.01	49.87
$\ell = 0.6$	47.89	49.56	**50.45**
$\ell = 0.7$	47.75	48.56	49.34

Key Observations

- The framework exhibits **robustness** to variations in ℓ and h, with Hit@1 fluctuations limited to a range of 2.7%.
- A **performance decline** is observed when the gap between ℓ and h becomes too large, potentially due to the exclusion of certain training queries that fall within the (h, ℓ) interval.
- A **moderate gap** between ℓ and h leads to slight performance improvements, suggesting that a balanced separation between positive and negative queries can enhance pattern differentiation without compromising the number of training queries.

The results indicate that setting $\ell = 0.6$ and $h = 0.5$ yields an improved Hit@1 score compared to the baseline reported in the original paper. Overall, this analysis underscores the robustness of the framework, which relies on a minimal set of hyperparameters, including ℓ, h, batch size b, and training epochs.

C Prompts

We keep only two prompt templates for our framework on all tasks: (1) The prompt template given to actor as initially instructions, and (2) the prompt template given to the comparator to conduct contrastive reasoning and generate the instructions for the actor. Below are the complete templates:

This is the prompt given to actor as initially instructions:

```
You are an expert user of a knowledge base, and your task is to answer a set of
    ↪ queries. I will provide your with the schema of this knowledge base:
<knowledge_base_schema>

You have access to several APIs that are pre-implemented for interaction with the
    ↪ knowledge base:
<func_call_description>

Information of queries: Below are several query examples that you need to carefully
    ↪ read through:
"
<example_queries>
"

Task: Given an input query, you should write the actions in Python code to calculate
    ↪  a ‘node_score_dict‘ for <n_init_candidates> node IDs, which are input as a
    ↪ list. These node IDs, referred to as ‘candidate_ids‘, are a subset of node
    ↪ IDs from the knowledge base, and the nodes belong to the type(s) <
    ↪ candidate_types>. In ‘node_score_dict: Dict[int, float]‘, each key should be
    ↪  a node ID, and each value should be the corresponding node score. This
    ↪ score should indicate the likelihood of the node being the correct answer to
    ↪  the query.

Output format: Firstly, you should establish a connection between the given queries
    ↪ and the query patterns to the schema of the knowledge base. Secondly,
    ↪ generate an outline for the code that will compute the scores for all the
    ↪ candidate nodes provided in the query examples. Finally, develop the main
    ↪ function named ‘get_node_score_dict‘, which takes two required parameters: ‘
    ↪ query‘ and ‘candidate_ids‘, and optional parameters declared in ‘
    ↪ parameter_dict‘. Note that ‘parameter_dict‘ is a dictionary of parameters
    ↪ and their default values where you can declare any parameters or weights
    ↪ used during computing the node scores. If no optional parameters are needed,
    ↪  leave ‘parameter_dict‘ as an empty dictionary. Overall, your output should
    ↪ follow the structure:

‘‘‘python
# <code outlines>
import <package1>
...

parameter_dict = {<parameter_name1>: <default_value1>,
                  <parameter_name2>: <default_value2>,
                  ...}

def get_node_score_dict(query, candidate_ids, **parameter_dict):
    node_score_dict = {}
    # your code
    return node_score_dict
‘‘‘

Hints:
- Observe the example queries carefully and consider the key attributes to extract.
- Use ‘‘‘python and ‘‘‘ to wrap the complete code, and do not use any other
    ↪ delimiters.
- You can use any of the pre-implemented APIs but should avoid modifying them.
- You can include other functions besides ‘get_node_score_dict‘, but ensure they are
    ↪  fully implemented.
```

```
- The code should be complete without placeholders and dummy functions.
- Optimize the integrity of the code, e.g., corner cases.
- Minimize computational expenses by early elimination of candidate nodes that don’t
    ↪  meet relational requirement (if any).
- Avoid conducting unnecessary and redundant computations, especially when using
    ↪ loops.
- Make use of ‘parameter_dict‘ to avoid hard-coding parameters and weights.
- Use the functions that end with ‘by_llm‘ wisely for more accurate searches.
- Use ‘debug_print‘ smartly to print out any informative intermediate results for
    ↪ debugging.
- Exclude or comment out any example uses of ‘get_node_score_dict‘ in the output
    ↪ code.

Your output:
```

This is the prompt given to comparator to generate the instructions for the actor:

```
<initial_prompt>

<previous_actions>

After executing the above actions on user queries, some queries have yielded good
    ↪ results, while others have not. Below are the queries along with their
    ↪ corresponding evaluation metrics:
Well-performing queries:
<positive_queries_and_metric>
Poorly-performing queries:
<negative_queries_and_metric>

Task:
(1) Firstly, identify and contrast the patterns of queries that have achieved good
    ↪ results with those that have not.
(2) Then, review the computational logic for any inconsistencies in the previous
    ↪ actions.
(3) Lastly, specify the modification that can lead to improved performance on the
    ↪ negative queries. You should focus on capturing the high-level pattern of
    ↪ the queries relevant to the knowledge base schema.
```

D Limitations

We identify several potential limitations of our work:

- **Scalability**: AvaTaR is designed to scale with large language models (LLMs) that support extended context lengths (up to 128k tokens), enabling it to handle numerous tools and complex tasks. However, increased latency and other practical limitations may hinder performance in scenarios requiring hundreds of tools or high complexity. Future research could focus on incorporating specialized, tool-augmented LLMs as auxiliary agents to facilitate smoother scaling.

- **Computation Requirements**: Managing longer contexts and multiple tool interactions within AvaTaR increases computational demands, which can significantly raise operational costs. These requirements necessitate substantial resources to maintain efficient performance, particularly when scaling to larger datasets or more intricate tasks.

- **Potential Failure Modes**: Although AvaTaR performs well on known queries, its performance may diminish when faced with queries that require new or unfamiliar combinations of tools. This limitation could be mitigated by integrating adaptive learning techniques and continuous monitoring, which would allow AvaTaR to better handle novel tool requirements.

E Function library

E.1 Complex Retrieval Tasks

Please refer to Table 8 and Table 9 for the detailed functions.

E.2 General QA Tasks

For general QA tasks, we use the following tools:

- `WEB_SEARCH`: A general-purpose tool that performs web searches to answer questions. Useful for retrieving up-to-date information from the internet when other sources are unavailable.
- `ARXIV_SEARCH`: This tool retrieves information about academic papers from Arxiv using a paper's unique ID. This function call can provide metadata and other details for academic references.
- `Wiki_SEARCH`: If you have a question or name to lookup, this tool uses a Wikipedia search to retrieve relevant information.
- `RETRIEVE_FROM_DB`: This tool is used to retrieve relevant information from a database. This is only available on ToolQA.

Function Name	Input	Output
ParseAttributeFromQuery	query: The string to be parsed, attributes: The list of attributes to be extracted from the query	This function parses a ‘query‘ into a dictionary based on the input list ‘attributes‘
GetTextEmbedding	string: The array of list to be embedded	Embeds N strings in a list into N tensors
GetRelevantChunk	query: The input query string, node_id: The ID of the node	Get the relevant chunk of information for the node based on the query
GetFullInfo	node_id: The ID of the node	Get the full information of the node with the specified ID
GetEntityDocuments	node_id: The ID of the node	Get the text information of the node with the specified ID
GetRelationInfo	node_id: The ID of the node	Get the relation information of the node with the specified ID
GetRelationDict	node_id: The ID of the node	Get the relation dictionary for the node with the specified ID, where the keys are relation type and values are neighbor nodes.
GetRelatedEntities	node_id: The ID of the node	Get the nodes related to the specified node
GetEntityIdsByType	type: The type of node to retrieve	Get the IDs of nodes with the specified type
GetEntityTypes	node_id: The ID of the node	Get the type of the node with the specified ID
GetEntityEmbedding	node_ids: An array of candidate node ids to be embedded	Get the embedding indices of nodes with ID ‘node_ids‘
ComputingEmbeddingSimilarity	embedding_1 and embedding_2	The cosine similarity score of two embeddings
ComputeQueryEntitySimilarity	query: The input query string, node_ids: An array of candidate node id to be compared with the query	Compute embedding similarity between ‘query‘ (str) and the nodes’ in ‘node_ids‘ (list)
ComputeExactMatchScore	string: The string to be matched, node_ids: The list of candidate node id to be compared with the string	For each node in ‘node_ids‘, compute the exact match score based on whether ‘string‘ is included in the information of the node
TokenMatchScore	string: The string to be matched, node_ids: The list of candidate node id to be compared with the string	For each node in ‘node_ids‘, computes recall scores between ‘string‘ and the full information of the node
SummarizeTextsByLLM	texts: The list of texts to be summarized	Use LLM to summarize the provided texts
ClassifyEntitiesByLLM	node_ids: The array of candidate node ids to be classified, classes: The list of classes to be classified into	Use LLM to classify each node specified by ‘node_ids‘ into one of the given ‘classes‘ or ’NA’
ClassifyByLLM	texts: The list of texts to be classified, classes: The list of classes to be classified into	Use LLM to classify each text into one of the given ‘classes‘ or ’NA’
ExtractRelevantInfoByLLM	texts: The list of texts to extract info from, extract_term: the terms to identify relevant information	Use LLM to extract relevant information from the texts based on extract_term, return sentences or ’NA’
CheckRequirementsByLLM	node_ids: The array of candidate node ids to be checked, requirement: The requirement to be checked	Use LLM to check if node(s) with ‘node_ids‘ satisfies to ‘requirement‘
GetSatisfictionScoreByLLM	node_ids: The array of candidate node ids to be scored, query: The input query from user	Use LLM to score the node with ‘node_ids‘ based on the given ‘query‘
FINISH	final_reranked_answer_list: The final answer	This function is used to indicate the end of the task

Table 8: Function library on STARK

Function Name	Input	Output
ParseAttributeFromQuery	query: The string to be parsed, attributes: The list of attributes to be extracted from the query	This function parses a ‘query‘ into a dictionary based on the input list ‘attributes‘
GetBagOfPhrases	image_ids: The image id array to get the phrases from	Returns a list of phrase list for each image in the image_ids list
GetEntityDocuments	image_ids: The image id array to get the text information from	Returns a list of text information for each image in the image_ids list
GetClipTextEmbedding	string: The list of strings to be embedded	Embed a string or list of N strings into N embeddings
GetPatchIdToPhraseDict	image_ids: The image list to get the patch_id to phrase list dictionary from	Returns a list of patch_id to phrase list dictionary for each image
GetImages	image_id_lst: The list of image ids	Return a list of images with corresponding ids
GetClipImageEmbedding	image_lst: The list of images to be embedded	Embed the images of a list of N image_ids into N tensors
GetImagePatchByPhraseId	image_id: the id of an image, patch_id: the patch id on the image	Return the patch image for the given image_id and patch_id
ComputingEmbeddingSimilarity	embedding_1 and embedding_2	The cosine similarity score of two embeddings
ComputeF1	string_to_match: The key word to be matched, strings: The list of strings to be calculated f1 score with the key word	Compute the F1 score based on the similarity between ‘string_to_match‘ and each string in ‘strings‘
TokenMatchScore	string_to_match: The key word to be matched, strings: The list of strings to be calculated recall score with the key word	Compute the recall score based on the similarity between ‘string_to_match‘ and each string in ‘strings‘
ComputeExactMatchScore	string_to_match: The key word to be matched, strings: The list of strings to be exact matched with the key word	Compute the exact match score based on whether ‘string_to_match‘ is exactly the same as each string in ‘strings‘
VqaByLLM	question: The question to be answered, image_lst: The list of images	Use LLM to answer the given ‘question‘ based on the image(s)
ExtractVisualAttributesByLLM	attribute_lst: The list of attributes to be extracted, image_lst: The list of images	Use LLM to extract attributes about the given ‘attribute_lst‘ from each image
FINISH	final_reranked_answer_list: The final answer	This function is used to indicate the end of the task

Table 9: Function library on Flickr30K Entities

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We design a novel and automatic framework that optimizes an LLM agent to effectively use the provided tools and make comprehensive analysis on the evolution of our key modules.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We did extensive survey on related work in the area of LLM agents, agent optimization, LLM agent for retrieval, and further discuss their limitations

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We elaborate the experiment details in the Experiment section including datasets, baselines, function libraries etc. We also release all the prompts we are using in the experiments for reproducibility.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our code and data are accessible at `https://anonymous.4open.science/r/AvaTaR-FBC4/`.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We include dataset information and training details in the Experiment part and Appendix. We also clearly describe the knowledge base and formally introduce the task settings.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: [NA]

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We run our experiments on a single NVIDIA A100-SXM4-80GB GPU and 32-core CPUs.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We do not induce any potential research harm mentioned in NeurIPS Code of Ethics in our paper.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discussed the impact in the introduction section.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our method provides a framework to better use LM but not releasing a LM.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Creators or original owners of assets mentioned in the paper are properly cited and the license and terms of use are respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.