REAPER: Reasoning based Retrieval Planning for Complex RAG Systems

Ashutosh Joshi*, Sheikh Muhammad Sarwar*, Samarth Varshney*,
Sreyashi Nag, Shrivats Agrawal, and Juhi Naik
(jashutos,smsarwar,varshsam,sreyanag,shrivagr,juhinaik)@amazon.com

ABSTRACT

Complex dialog systems often use retrieved evidence to facilitate factual responses. Such RAG (Retrieval Augmented Generation) systems retrieve from massive heterogeneous data stores that are usually architected as multiple indexes or APIs instead of a single monolithic source. For a given query, relevant evidence needs to be retrieved from one or a small subset of possible retrieval sources. Complex queries can even require multi-step retrieval. For example, a conversational agent on a retail site answering customer questions about past orders will need to retrieve the appropriate customer order first and then the evidence relevant to the customer's question in the context of the ordered product. Most RAG Agents handle such Chain-of-Thought (CoT) tasks by interleaving reasoning and retrieval steps. However, each reasoning step directly adds to the latency of the system. For large models this latency cost is significant – in the order of multiple seconds. Multi-agent systems may classify the query to a single Agent associated with a retrieval source, though this means that a (small) classification model dictates the performance of a large language model. In this work we present REAPER (**REA**soning-based **P**lann**ER**) - an LLM based planner to generate retrieval plans in conversational systems. We show significant gains in latency over Agent-based systems and are able to scale easily to new and unseen use cases as compared to classification-based planning. Though our method can be applied to any RAG system, we show our results in the context of a conversational shopping assistant.

ACM Reference Format:
Ashutosh Joshi*, Sheikh Muhammad Sarwar*, Samarth Varshney*, and Sreyashi Nag, Shrivats Agrawal, and Juhi Naik. 2024. REAPER: Reasoning based Retrieval Planning for Complex RAG Systems. In *Proceedings of CIKM 2024: Conference on Information and Knowledge Management (CIKM '24).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Conversational shopping assistants help customers navigate their shopping journey by providing relevant information at the right time. They are equipped to answer customer questions on shopping needs, products, comparisons, make recommendations based on this context, and facilitate product discovery. A conversational shopping assistant is thus trained on both product catalog and open data sources. It uses a RAG (Retrieval Augmented Generation) framework [15] where the response to a customer's query is generated by an LLM, using evidence from one or more retrieval sources. Most complex dialog systems cover a large variety of topics. They need to retrieve evidence from data stores and indexes that are potentially petabytes in size and store heterogeneous documents in multiple modalities. These massive data stores are usually structured as multiple homogeneous indexes rather than a single monolith. For efficient retrieval, the dialog system needs to decide which indexes to query and even when to let the LLM to answer through its own knowledge without relying on retrieved evidence.

Retail conversational shopping assistants need to retrieve evidence from multiple sources like reviews, product information, help pages, delivery information and more. These sources can include a mix of classical retrieval stores like HNSW [17] indexes built using encoder models [12, 27], API's that link to internal or external services (eg: an API to get assembly instructions from a manufacturer's site) or answers using pre-trained knowledge.

Each retrieval source or retriever has associated latency and hardware costs. Thus, dialog systems need to decide which retrievers to invoke for a given query. The situation is further complicated when the retrieval itself can become multi-step. Figure 1 shows a scenario where changing the query from *"How much memory is on **a** Galaxy phone"* to *"How much memory is on **my** Galaxy phone"* significantly changes the retrieval plan. In the first case, the retrieved evidence comes from information about Galaxy phones in general. Any sufficiently large generic LLM will be able to answer this from its pre-trained knowledge. For the second question though, we first need to identify the exact phone that was purchased by the customer and give specific information pertaining to that phone.

LLM Agents [21, 26] are able to handle the retrieval complexity by interleaving retrieval and reasoning calls. However, each reasoning step directly adds to the latency of the systems. For large models this latency cost is in the order of multiple seconds. Multi-agent systems [3, 4] use classifiers to route the query to an appropriate agent (*question-pairing*) or use multiple agents to generate candidate responses and a final agent to select the best response (*response-pairing*). Question-pairing gates a powerful LLM using a classifier and thus can limit the ability of the LLM, while response-pairing adds complexity, hardware costs and latency to the system by requiring multiple LLMs to process the query in order to generate a response.

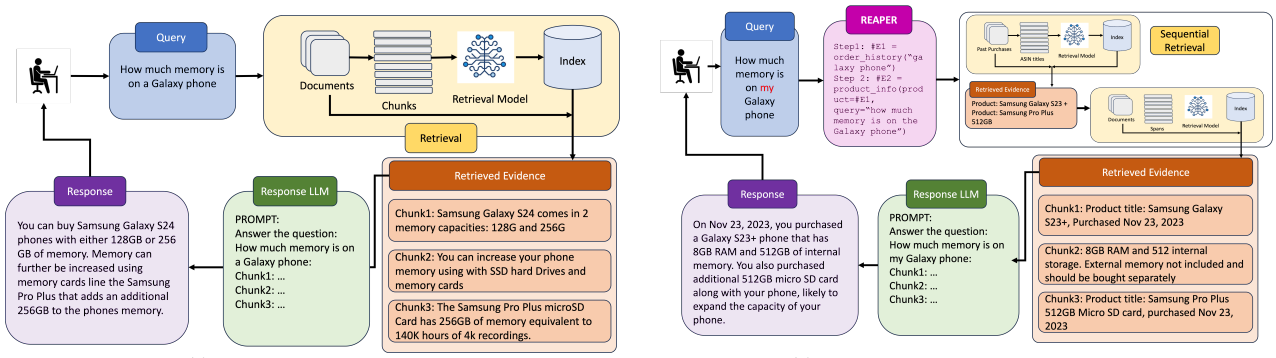
1.1 Our Contribution

In this paper, we introduce REAPER – a **REA**soning based **P**lann**ER** – for efficient retrieval required for complex queries. Using a single

*these authors contributed equally.

CIKM '24, October 21 – 25, 2024, Boise, ID

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX



(a) Single-Step Retrieval in RAG **(b) Multi-Step Sequential Retrieval for RAG**

Figure 1: Traditional RAG systems rely on retrieving evidence in parallel from one or more sources. Conversational shopping can include features like personalization (questions about past purchases, preferences, subscriptions, etc), shopping recommendations, and more that require multi-step retrieval. These use cases can be complex enough that either an Agent is required to identify the steps, or retrieval needs its own CoT planner. We introduce REAPER for CoT retrieval planning

and much smaller LLM, REAPER generates a plan that includes the tools[1] to call, the order in which they should be called and the arguments to each tool. By generating the entire retrieval plan in a single step and using a smaller LLM, we are able to minimize the latency cost as compared to single- or multi-agent systems and still maintain the response quality. REAPER achieves 96% accuracy when selecting the right tool sequence and 92% accuracy on generating the correct tool arguments. We also show that compared to classification based question-pairing systems, REAPER is able to easily scale to new retrieval sources (tools) with very little training data and to new use cases using the current tools with just in-context examples.

Though our architecture follows the mold of multi-agent systems, it does not implement communication between the LLMs, which is the key element of such systems. Hence, we consider REAPER a stand-alone planner rather than a multi-agent system. In this paper, we keep the response generation LLM constant and focus on the retrieval planning capabilities of REAPER rather than the response quality, with the understanding that with better evidence retrieval the response LLM will generate a better answer.

2 LITERATURE REVIEW

Open Domain Question Answering is the task of accurately answering a query by *retrieving* relevant documents, and interpreting them via a *reader*. Extractive Readers predict an answer span from the retrieved documents [12, 19]. Generative Readers generate answers in natural language using sequence-to-sequence models [8, 23]. With the advent of LLMs, retrieval augmented generation (RAG) has gained popularity Gao et al. [7]. Almost all of RAG research focuses on how retrieved evidence can be used to improve some quality metric of the generated response [2, 9, 13, 14].

Multi-Hop QA (MHQA) requires a model to *reason* over several steps and retrieved evidences to reach an answer. Similar to RAG, MHQA research focuses on how LLMs use evidence rather than how to retrieve the correct evidence [18]. For example, Khattab et al. [13] and Yao et al. [26] tackle complex Chain-of-Thought (CoT) reasoning by interleaving retrieval and reasoning steps in different ways. On the other hand, Xu et al. [24] introduce REWOO (Reasoning WithOut Observation), in which they argue that generating the complete plan in a single step and then executing it allows for more accurate planning. In a real-world application like conversational shopping assistants, completing the planning in a single step can help reduce the overall latency by limiting the LLM calls.

In MHQA and other Chain-of-Thought (CoT) reasoning tasks, retrieval is the main bottleneck [18]. Very few prior work, though, consider the problem of efficient retrieval in RAG, MHQA or CoT systems. Even multi-Agent systems that consider retrieving the most relevant evidence, focus on answer quality instead of efficiency. Fang et al. [4] use an LLM to route the customer query to a one of three LLMs trained to either *chit-chat*, *recommend a product* or *ask a question*. Multi-step retrieval, though, would still require multiple calls to the system. Clarke et al. [3] use an LLM to select from responses of several Agents (*response-pairing*). They also compare it to an approach of using a different classifiers to route the query to a single LLM specialized for that query shape (*question-pairing*). They find that the response-pairing generates better answers but adds complexity, while question-pairing is faster and cheaper but has lower answer quality. Jeong et al. [10] propose Adaptive-RAG, where they improve RAG time by teaching a smaller LLM to dynamically decide to use no-evidence, single-step RAG or CoT. with interleaved reasoning and retrieval steps.

REAPER combines concepts from Adaptive-RAG, question-pairing and ReWOO. We propose an architecture, where REAPER – a smaller LLM – generates the retrieval plan via CoT reasoning, and a large LLM uses the evidence to generate the appropriate response. An exemplar system diagram is shown in Figure 1.

[1]Borrowing from Agent literature, we treat retrievers as tools. However, we also invoke tools that perform supplementary tasks like time conversions. Thus, tools are a superset of retrievers.

3 PROBLEM STATEMENT

Our objective is to allow conversational systems to scale to queries requiring CoT retrieval plans (both single and multi-step) and to new use cases without incurring the high latency and hardware cost of an Agent LLM, in a data efficient manner. We do this by moving CoT reasoning specific to retrieval to a specialized, smaller LLM. This REAsoning based PlannER (REAPER) takes customer query and contextual information as input. Figure 1(b) shows an example of a query requiring multi-step retrieval. For a conversational shopping assistant, a popular use-case is queries about products. Thus, when available, we provide the product information as context to REAPER. Figure 2 shows example plans where the user can ask a question with or without product context. Based on the conversational system, the context can be extended to other information like conversational history, date/time at which the question is asked, user information, url or identifier of the page on which the question is asked, etc. To generate retrieval plans, we require REAPER to:

- Understand all the available tools used for generating evidence.
- Generate a retrieval plan that can work for no-evidence-retrieval, single-step retrieval and multi-step retrieval. Since REAPER will likely be the ingress point into the conversational system, the plans for all of these should be generated using the same prompt.
- Since REAPER mistakes can propagate all the way to the ultimate response, REAPER needs to achieve high accuracy in tool selection, sequencing and format and arguments of the tools.
- For latency and hardware gains, the REAPER LLM should be significantly smaller than the answer generation LLM of the conversational system.
- REAPER should be scalable to new retrievers or tools with minimal new data and training.
- REAPER should not hallucinate new tools for use cases it has not seen before and should be able to follow changes in the tool collection. Thus, it needs to retain good instruction following ability, although high performance on general-purpose instruction following is not required since the objective is to use it for the sole task of retrieval planning.

3.1 Comparable Systems

The most common architectures for conversational systems include an agent [5, 20] or a classification system that helps route the queries [3, 4, 10]. We thus use our implementations of such systems as baselines. We simulate a conversational Agent (or multi-Agent) by sequentially calling Claude-Sonnet [1] for identifying the steps in a multi-step retrieval plan. The number of retrieval-related calls is equal to the number of steps in the retrieval plan.

As the classification-based baseline (question-paring multi-Agent system [3]), we trained an ensemble of 2 RoBERTa models to classify queries into six classes. Our ensemble achieves better performance than a single classifier and thus, is a stronger baseline. We use a total of 150K queries to fine-tune the Roberta models. However, to add a new retriever (new class) we will need to collect tens of thousands of representative queries which is expensive and time-intensive.

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "how do i cancel all subscribe and save items"
# Answer:
Step 1: Ask customer support - #E1 = customer_support("how do i cancel all subscribe
and save items")
```

```
# Input:
Context: {"product_id": "J6FLBNBQ31", "product_title": "MICHELIN Defender LTX M/S
All-Season Radial Car Tire for SUVs and Crossovers, 235/65R18 106T"}
Question: "is the tread depth between 0.15 inches and 0.25 inches"
# Answer:
Step 1: Fetches specific information for a product relevant to a particular attribute
- #E1 = prod_qna("J6FLBNBQ31", "What is the tread depth of the MICHELIN Defender LTX
M/S All-Season Radial Car Tire?")
```

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "show me some options for nut free chocolate coins"
# Answer:
Step 1: Fetch product search results - #E1 = prod_search("nut free chocolate coins")
```

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "What time will my sheet pan arrive tomorrow?"
# Answer:
Step 1: Fetch the date range - #E1 = date_math("tomorrow")
Step 2: Fetch shipment status - #E2 = shipment_status(date_range=#E1,
        keywords="sheet pan")
```

Figure 2: Example REAPER plans. Note that REAPER is able to incorporate context (second plan) and generate multi-step retrieval plans when necessary (last plan).

Following classification, we use the Mistral 7B LLM (instruct-v0.2 version) for generating appropriate arguments for the retrievers or APIs. Complex multi-step retrieval is handled by assigning customer queries that need multi-step retrieval to a separate class which then initiates a static multi-step workflow. This means that for some such workflows we may need to call the Mistral LLM multiple times with the appropriate prompt that generates the arguments for the particular retriever. We also note that as the number and complexity of queries grows the classification based approach becomes cumbersome and does not scale. The number of classes with multi-step retrieval also grows combinatorially with the number of retrievers.

We aim for REAPER to match or beat RoBERTa ensemble performance while also developing new capabilities like training-data efficiency, dealing with ambiguities and complex retrieval cases.

For a fair comparison, we evaluate within the strengths of the classification models. Thus, we have limited the number of retrieval classes to a small number (six), followed by a call to the Mistral model to generate all arguments needed in the workflow for the class. Thus, REAPER has a harder task of generating the plan along with the right arguments, while the classification system simply needs to classify the queries to one of six classes.

4 REAPER

To develop an LLM that meets the requirements of Section 3, we need a reasonably small LLM with strong instruction following abilities. We use Mistral-7B-Instruct-v0.2 [11] based on its performance on open IFT benchmarks. However, even with significant prompt tuning and in-context examples, the Mistral model was prone to hallucinations (see Figure 3 for examples). Hence, we fine-tune the model for our use case. In this section, we explain our methodology for designing the REAPER prompt and selecting fine-tuning data to maintain instruction following ability and eliminate hallucinations while learning the specialized task of retrieval tool planning.

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "how do i cancel all subscribe and save items"
# Answer:
Step 1: Ask customer support for cancelling all subscribe and save items -
#E1 = customer_support("how do I cancel all subscribe and save items")
```

```
# Input:
Context: {"product_id": "J6FLBNBQ31", "product_title": "MICHELIN Defender LTX M/S
All-Season Radial Car Tire for SUVs and Crossovers, 235/65R18 106T"}
Question: "is the tread depth between 0.15 inches and 0.25 inches"
# Answer:
Step 1: Retrieve information regarding the tire -
#E1 = prod_qna("Q1EOAR1GRT", "What is the tread depth of MICHELIN Defender LTX
M/S All-Season Radial Car Tire for SUVs and Crossovers, 235/65R18 106T?")
Step 2: Check tread depth -
#E2 = decontextualizer("#E1", "What is the tread depth of the tire?")
Step 3: Compare tread depth with given range -
#E3 = prod_qna("Q1EOAR1GRT", "Does the tread depth of MICHELIN Defender LTX
M/S All-Season Radial Car Tire for SUVs and Crossovers, 235/65R18 106T fall
between 0.15 inches and 0.25 inches?")
Step 4: Compare #E2 with #E3 -
If #E2 = #E3 then "Yes, the tread depth is between 0.15 inches and 0.25 inches"
else "No, the tread depth is not between 0.15 inches and 0.25 inches"
```

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "show me some options for nut free chocolate coins"
# Answer:
Step 1: Fetch product search results - #E1 = prod_search("nut free chocolate coins")
```

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "What time will my sheet pan arrive tomorrow?"
# Answer:
Step 1: Fetch the date range - #E1 = date_math("tomorrow")
Step 2: Fetch shipment status - #E2 = shipment_status(keywords ="sheet pan",
                                                      date_range=#E1)
Step 3: Extract the estimated delivery time from the shipment status -
                          #E3 = decontextualizer(#E2, "Estimated delivery time:")
Step 4: Convert the estimated delivery time to a readable format -
                          #E4 = date_math(#E3, "format_date_time", "%I:%M %p")

# Output:
[["date_math", "tomorrow"], ["shipment_status", "sheet pan"],
["decontextualizer", ""], ["date_math", "format_date_time"], [""]]
```

Figure 3: Mistral plan examples. For simple queries (first and third) Mistral produces the right plans. However, for queries requiring multi-step retrieval, Mistral goes beyond retrieval evidence generation, hallucinating steps like Compare (second plan) and Extract estimated delivery time (fourth plan).

4.1 REAPER Prompt Design

An LLM prompt p consists of an input x, an instruction set I, and a set of m in-context examples, $E = \{(\tilde{x}_1, \tilde{y}_1), (\tilde{x}_2, \tilde{y}_2), \ldots, (\tilde{x}_m, \tilde{y}_m)\}$ that help the model understand the desired task.

In our prompt, I includes instructions like the role of the LLM as well as all the tools $T = \{\tilde{f}_1, \tilde{f}_2, \ldots, \tilde{f}_t\}$. The tools are essentially API calls that the REAPER LLM needs to understand. We provide the tool name, tool signature, its natural language description and an example usage in the prompt. Finally, I contains task instructions and constraints. Exemplar elements of the prompt are shown here:

```
### Role:
You are an AI assistant to a salesperson at a big retail store. Your
goal is to find the right information to help the salesperson answer
the customer's question.

### System Instruction:
Your goal is to generate a step by step plan using the tools listed
below to get the information needed to answer a customer question.
The output of one tool can be fed to another in a sequential manner.
Each step may use only one tool. Some parameters of a tool can be
generated with help of provided capabilites.

The set of the candidate tools, their definitions, example usages are:

1. prod_qna - Tool: Fetches specific information for a particular
aspect or attribute of the product . It needs a product ID and a
query as input.
...
```

For our case, the input $x = \{q, c\}$ includes the customer query, q and page context, c. A customer can reference information gathered during their shopping journey. So on a product Detail Page (DP), we include the product title in c. On non-product pages, like Search Results Page, Landing Page, Checkout, etc., c is empty. Page context is necessary for anaphoric and contextual de-referencing. For example, when a customer asks *"What is your favorite color"* when say, they open the shopping app, they are simply engaging in small talk with the conversational system. However, if they ask the same question on the DP of say a t-shirt that is available in multiple colors, they expect the response in the context of that t-shirt. In former (small-talk) case, the LLM may wish to answer with no evidence along the lines of *"I am an AI assistant and do not have favorite colors"* (paraphrased for brevity). In the latter case, it will use evidence from reviews to deduce the popular colors or sentiment around different colors to form an answer.

4.2 REAPER Data Generation Approach

For REAPER we aim to balance two contrasting objectives: 1) the model needs to generate the plan for an in-domain task in a precise format with the right tool signatures, 2) the model should be able to understand changes in the input including nuanced changes in instructions and tools and adapt the plans accordingly. We can meet the first requirement by training the model with a large number of precise REAPER plans. However, this causes the model to overfit on the planning task and it loses its instruction following and reasoning capability. On the flip side, without enough plans in the training data, we see that the model tends to hallucinate responses.

An important design consideration for fine-tuning an LLM for instruction following is to provide it with a diverse set of prompt and output pairs so that it does not overfit on a specific task template and catastrophically forget its ability to closely follow instructions. This becomes particularly challenging since our primary task is retrieval planning and so the scope of introducing diversity in the instruction set is limited. We develop the following modules for different aspects of input and output data diversity.

4.2.1 Tool Evolve (TEvo): Evolution of Base Tool Prompt. We introduce a novel module **Tool Evolve (TEvo)** that takes the tool prompt as input and produces a semantically similar prompt in a way that the output, y does not change. The technique is similar to introducing adversarial noise into images for building robust image classifiers [16]. To force our model to pay attention to tools and their corresponding descriptions, we select the tool(s) required to for a particular query and a random subset of the remaining tools to include in the instruction section of our prompt. We also create a pool of name variation and description paraphrases for each tool and sample from these. For example, to answer questions about a product, our tool names can be `prod_qna`, `product_information`, `product_facts`, etc. Finally, we also vary in-context examples by sampling from a small pool of human generated plans.

4.2.2 Tool-Task Generator (TTG). In Wizardlm [25], the authors increase the complexity of simple tasks and add these tasks to the IFT training data in order to improve the instruction following capability of an LLM. Similarly, we introduce an approach to create diversified tasks related to retrieval planning. This forces the LLM to understand tools and retrieval plans.

Given a primary task of generating a retrieval plan $T_{primary} = (x, y)$, where $x = \{q, c\}$ is the input containing the query q and context c, we transform the task into multiple related tasks.

$$T_i = f_i(T_{primary}) \tag{1}$$

Here, f_i is our proposed task-specific transformation function that creates *seven* secondary tasks to provide the fine-tuned model a diverse set of capabilities related to planning as below:

(1) Generate q from a plan y: $T_1 = \text{generate\_query}(y)$
(2) Complete partial plan: $T_2 = \text{complete\_partial\_plan}(x, y_{\text{partial}})$
(3) Identify the correct tools: $T_3 = \text{identify\_tools}(x)$
(4) Generate the masked step of a plan: $T_4 = \text{complete\_masked\_step}(x, y_{\text{masked\_step}})$
(5) Reorder randomly shuffled plan steps in correct sequence: $T_5 = \text{reorder\_steps}(x, y_{\text{shuffled}})$,
(6) Generate masked value of a parameter: $T_6 = \text{identify\_masked\_params}(x, y_{\text{masked\_param}})$
(7) Limit tools in plan to the ones specified in the prompt, p: $T_7 = \text{use\_provided\_tools}(x, p)$

Thus, the TTG module creates seven diverse (prompt, output) pairs from (x, y) by applying secondary task transformations. We then sample from this diverse set of tasks to fine-tune the REAPER, with the aim of developing a robust understanding of the overall task structure and enhancing its ability to generate retrieval plans for complex customer queries.

4.2.3 Diverse Query Sampler (DQS). In addition to adding diversity in prompt using TEvo and diversity in output using TTG, we also diversify the input x, which is a question from a customer with a page context. Similar queries cause the model to fixate on particular query shapes leading to performance degradation when the model encounters out-of-distribution cases. To this end, we propose **Diverse Query Sampler (DQS)** that automatically generates a sample of customer queries that are semantically dissimilar. We obtain human annotations of the diverse samples with relevant tools and parameters based on the conversational context.

Given a high-quality curated initial (small) pool of n customer queries, $Q_{\text{initial}} = \{q_1, q_2, \ldots, q_n\}$ and a larger pool of generated or sampled customer queries Q_{large}, DQS introduces diversity by:

1. Generating BERT-based embeddings $\mathbf{e}_i$ for each query, $q_i \in Q_{\text{large}}$, where $\mathbf{e}_i = \text{RoBERTa}(q_i)$, and taking the same step for queries in Q_{initial}. The RoBERTa model embedding space was optimized using the labeled samples in Q_{initial}.

2. Calculating the pairwise cosine similarity between the query embeddings in Q_{initial} and Q_{large} to obtain a similarity matrix S, where $S[i, j] = \cos(\mathbf{e}_i, \mathbf{e}_j)$ for $q_i \in Q_{\text{initial}}$ and $q_j \in Q_{large}$.

3. Identifying the most similar and most dissimilar pairs of queries in Q_{large} based on the similarity matrix S. Let these be the set of queries at the extremes of diversity, denoted as Q_{extreme}.

4. Removing Q_{extreme} from Q_{large} to obtain a refined pool, $Q_{\text{refined}} = Q_{\text{large}} \setminus Q_{\text{extreme}}$.

5. Randomly sampling a subset of queries from the larger pool Q_{large} to obtain the final diverse set of customer queries Q_{diverse}. The size of Q_{diverse} is chosen such that $|Q_{\text{diverse}}| = |Q_{\text{initial}}|$.

The goal of this process is to ensure that the queries in $\mathbf{Q}_{\text{diverse}}$ have a balanced representation of semantic diversity, reducing model bias and enhancing the ability of REAPER to generalize across different customer information needs.

4.2.4 General purpose IFT data. To retain the model's instruction following ability, we include general purpose instruction fine-tuning datasets in addition to the REAPER tool planning data. We use ShareClaude and open-source tool usage data from ToolAlpaca [22] for generic IFT data. We call this dataset Generic-IFT . To further enhance the model's ability to follow nuanced changes in the input instruction, we utilize a framework inspired from Evol-Instruct [25] that automatically generates more complex IFT data by adding constraints on simple instructions and samples to maintain a roughly equal proportion of query complexity as measured by the prompt length used for the task. We call this dataset Generic-IFT-Evolve. Our final fine-tuning dataset, REAPER-IFT is a combination Generic-IFT/Generic-IFT-Evolve and our tool-annotated queries which are diversified using TEvo, TTG and DQS.

5 EXPERIMENTS

In this section we present experimental setting and results comparing REAPER against Mistral-7B-Instruct-v0.2 , Claude3-Sonnet and the our ensemble classifier described in Section 3.1. Our experiments show the need for fine-tuning for the plan generation as larger models like Claude3-Sonnet or Mistral tend to hallucinate. We also present ablation studies to show the impact of different components that we proposed to create our training data.

5.1 Comparison with Classifier-based Planners

Our ensemble classifier is described in Section 3.1. We evaluate the models along two dimensions:

(1) **Tool Selection:** Given a query, we manually evaluate if the model selects the correct tool(s) in the proper sequence to retrieve evidence. As multi-step retrieval is just another class in the question-pairing system, this metric can be directly compared to the classification metrics in question-pairing. We present accuracy, precision, recall and F1-score for this evaluation.
(2) **Argument Extraction:** The other aspect of REAPER is the accuracy of the arguments fed to the tools (refer Figure 2). In the current setting, only two tools `prod_search` and `shipment_status` require the arguments different from the customer query. We thus restrict our evaluation to plans involving these tools. Our baseline system uses the Mistral-7B-Instruct-v0.2 model to generate the arguments using a prompt specifically tuned for each class. REAPER does both tool selection and argument generation for all the tools in a single prompt.

Table 1 shows the training data size and precision, recall and F1 metrics for REAPER, the baseline ensemble and Mistral-7B-Instruct-v0.2 with only in-context tuning. Given Mistral tool accuracy is so low, we did not compute the Argument accuracy for it.

Table 1: Comparison of REAPER performance with Baselines on six retrieval classes

	Mistral (No fine-tuning)			Ensemble Classifier				REAPER			
Classes	P	R	F1	P	R	F1	#Training Examples	P	R	F1	#Training Examples
Customer Support	90	61	73	95	81	88	24621	95	94	94	1127
Shipment Status	97	72	83	96	96	96	16150	98	94	96	996
Product Search	82	65	72	84	99	91	38683	91	100	95	1289
Product QnA	47	80	59	98	97	98	30813	93	99	96	1045
Review Summary	79	93	85	99	96	97	9875	100	94	97	594
General Knowledge	85	67	75	98	99	99	35934	100	93	96	1245
Tool Accuracy	72%			94%				**96%**			
Argument Accuracy	–			88%				**92%**			

5.1.1 Evaluation Datasets. Since conversational shopping is new, traffic distributions are still skewed towards existing traffic patterns instead of the new conversational use cases. So instead of sampling traffic to generate our evaluation set, we use a balanced evaluation set of 600 queries such that the corresponding plans have a roughly equal proportion of tools (classes)

5.2 Comparison with Open Models

It is generally desirable to solve a task using a language model using in-context learning. Despite several weeks worth of effort, we could not prompt-tune Mistral-7B-Instruct-v0.2 to reach the target performance. Figure 3 shows examples of the plans generated by Mistral-7B-Instruct-v0.2 with prompt tuning. We also tested out our REAPER prompt on Claude3-Sonnet (Figure 4) and could not prevent hallucinations. In addition, Claude3-Sonnet latency was **~2s** per step as compared to **207ms** for the entire plan for REAPER and Mistral models – introducing an order of magnitude latency cost in addition to more powerful hardware required to run Sonnet.

Alongside open-weight general instruction fine-tuned models such as Mistral (instruct v0.2), we evaluated REAPER against ReWOO-LLaMA-7B (23% accuracy) which is fine-tuned for tool selection, and against a closed-weight model GPT-3.5-turbo (82.5% accuracy). Given that GPT-3.5-turbo is not able to outperform REAPER out-of-the-box, our assumption of developing a specialized planning model [6] is justified. ReWOO-LLaMa-7B is distilled from GPT-3.5 in a supervised manner, and as a result it performs poorly on unseen tasks. This indicates the need for maintaining the instruction following capability of the planner to allow new use-cases easily.

5.3 Ablation Study of Data Components

We show the effectiveness of TEvo, DQS and IFT-Evolve in Table 2. Apart from tool selection accuracy, we also investigate how these components help the model to follow instructions by introducing an adversarial prompt where we remove the `prod_qna` tool and obtain REAPER predictions on the test dataset. Since the model is instructed to use only the specified tools it should not use `prod_qna` to generate a plan. We measure instruction following capability [2] as (1-proportion of plans that use `prod_qna`).

[2]We plan to evaluate performance on open IFT benchmarks, though we expect the performance to be similar to our domain specific evaluation

We observe that REAPER including all three components achieves both the highest tool selection accuracy and the best instruction following as none of its plans in the adversarial setting uses the omitted `prod_qna` tool. When we remove Generic-IFT-Evolve data from training and use Generic-IFT instead, tool selection accuracy

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "how do i cancel all subscribe and save items"
# Answer:
Here is a step-by-step plan to answer the question
"how do i cancel all subscribe and save items":
Step 1: Ask customer support -
#E1 = customer_support("how do i cancel all subscribe and save items")
The customer support tool can fetch the relevant help page from Amazon.com
to provide instructions on canceling Subscribe and Save subscriptions.
```

```
# Input:
Context: {"product_id": "J6FLBNBQ31", "product_title": "MICHELIN Defender LTX M/S
All-Season Radial Car Tire for SUVs and Crossovers, 235/65R18 106T"}
Question: "is the tread depth between 0.15 inches and 0.25 inches"
# Answer:
Step 1: Retrieve information regarding the product tread depth -
#E1 = prod_qna("J6FLBNBQ31","What is the tread depth of product J6FLBNBQ31 MICHELIN
Defender LTX M/S All-Season Radial Car Tire for SUVs and Crossovers,
235/65R18 106T?")
Step 2: Check if tread depth is between 0.15 and 0.25 inches -
#E2 = general_knowledge("Is tread depth #E1 between 0.15 and 0.25 inches?")
```

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "show me some options for nut free chocolate coins"
# Answer:
Here is a step by step plan to find information to answer the customer's
question using the available tools:
Step 1: Perform a product search - #E1 = prod_search("nut free chocolate coins")
Step 2: Get review summary for top search results -
#E2 = review_summ(<top product ID from #E1>)
Step 3: If review summary does not provide enough information, retrieve additional
details - #E3 = prod_qna(<top product ID from #E1>,
"What are the ingredients and allergen information for this product?")
This plan first searches for nut-free chocolate coin products on Amazon using
the prod_search tool. It then retrieves review summaries for the top results to
see what customers are saying about being nut-free using review_summ.
If more specific ingredient and allergen details are needed, it uses prod_qna to get
that information directly for the top product.
```

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "What time will my sheet pan arrive tomorrow?"
# Answer:
Step 1: Fetch the date range for tomorrow - #E1 = date_math("tomorrow")
Step 2: Fetch shipment status for "sheet pan" orders within the date range
        #E1 - #E2 = shipment_status(keywords="sheet pan", date_range=#E1)
```

Figure 4: Example Claude3-Sonnet plans. Similar to Mistral, we also see hallucinations in plans. For example, extra step in the second plan and extra explanation in the third plan. With more effort it may be possible to reduce hallucinations. Claude3-Sonnet latency though, is 10x of Mistral latency.

drops. The model also loses instruction following capability as it uses `prod_qna` in 24% of plans. We observe the same phenomenon when we remove the TEvo (task diversity) and DQS (input query diversity) components from training, with a marked drop in accuracy when we drop DQS. This suggests that TEvo (task diversity) is needed for instruction following capabilities and DQS is needed for the model to understand the different query shapes.

5.4 Effect of Training Data Proportions

Table 3 shows the different data proportions in the training set. We found that it is essential to either add ShareClaude (Generic-IFT) or its evolved version – Generic-IFT-Evolve with REAPER plans in the training set. Otherwise the model overfits to the REAPER use case and catastrophically forgets its instruction following capability. In this case, we see 77% of queries hallucinate tools that the model has seen in training but are not in the instruction (IFT score = 0.23). With only generic-IFT training data, the model achieves only 20% accuracy for the in-domain task of tool selection.

The proportion of the generic IFT training data does affect the performance of the model. In the middle three rows of Table 3, we varied the proportion of the Generic-IFT-Evolve data from 40% to 60%, while the last three rows show variation in the number of REAPER planning tasks (cf. Section 4.2.2) per query. We see improvement in both metrics as we increase the amount of IFT data and REAPER plans up to a point, beyond which we see a drop in tool selection accuracy. We found the best balance between accuracy and instruction following is seen in a roughly 1:6 proportion of REAPER tasks and generic IFT (Generic-IFT-Evolve (100%) + TTG (4-task)). However, we expect to tune these hyperparameters as we expand the system to new retrievers and use cases.

5.5 Analysis on Generalization

Can REAPER generate a novel plan? Our training data has multi-step plans only in the order status context. Even so, using in-context examples alone REAPER is able to generate the right tools sequences

Table 2: Impact of training data selection components

Model	Tool Accuracy	Instruction Following
REAPER	0.96	1.00
REAPER w/o Generic-IFT-Evolve	0.95	0.76
REAPER w/o Generic-IFT-Evolve and DQS	0.87	1.00
REAPER w/o Generic-IFT-Evolve, TEvo and DQS	0.91	0.85

Table 3: REAPER Data Proportion Study. Note that the REAPER tasks are generated using 6K unique in-domain queries, while the Generic IFT data is from open datasets.

Training data mix	Tool Accuracy	#REAPER tasks	#Generic IFT	Instruction following
Generic-IFT-Evolve (0%)+ TTG (3-task)	0.92	19k	0k	0.23
Generic-IFT-Evolve (100%)+ TTG (0-task)	0.2	0k	147k	0.71
Generic-IFT-Evolve (40%)+ TTG (3-task)	0.93	19k	60k	1.00
Generic-IFT-Evolve (50%)+ TTG (3-task)	0.93	19k	75k	1.00
Generic-IFT-Evolve (60%) + TTG (3-task)	0.9	19k	90k	1.00
Generic-IFT-Evolve (100%) + TTG (3-task)	0.86	19k	147k	1.00
Generic-IFT-Evolve (100%) + TTG (4-task)	0.95	25k	147k	1.00
Generic-IFT-Evolve (100%) + TTG (5-task)	0.94	32k	147k	1.00

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "Which battery is compatible with my camera that I ordered yesterday"
# Answer:
Step 1: Fetch the date range - #E1 = date_math("yesterday")
Step 2: Fetch shipement status - #E2 = shipment_status(keywords="camera",
        date_range=#E1)
Step 3: Fetch information regarding the camera battery - #E3 = prod_qna("B09GW671M2",
"What battery is compatible with the camera B09GW671M2 that was ordered yesterday?")
```

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "Is my desk made of real wood?"
# Answer:
Step 1: Fetch shipement status - #E1 = shipment_status(keywords="desk")
Step 2: Retrieve information regarding desk material - #E2 = prod_qna(#E1,
        "Is my desk made of real wood?")
```

Figure 5: Plans for new use cases. The first plan hallucinates the product_id but the second plan shows no hallucinations after we add just 25 post-purchase examples to training set

```
# Input:
Context: {"product_id": "", "product_title": ""}
Question: "ink cartridge for HP 4650"
# Answer:
Step 1: Fetch product search results - #E1 = compatible_products(base_product_id=
"B013SKI4X8", compatible_product_type = "ink catridge")
```

Figure 6: REAPER plans with tools not in training data

for query shapes it has never seen before. We did see occasional hallucinations. We added only 25 more diverse multi-step examples and REAPER was able to generate accurate brand new plans without hallucinations. Figure 5 shows some examples.

Can REAPER be fine-tuned with limited data? A practical use-case of REAPER is adapting to a new retrieval source with limited amount of data. We tested this, by adding a new `no-evidence` class called `human_small_talk`. We added only 286 examples of this class to our training data and saw that the model achieves an F1 score of 0.92 which comparable performance to other sources.

Can REAPER use a novel tool? We explore how well REAPER can learn to use a new tool without any training data at all and based solely on in-context examples for the tool. To do this, we added a new toolcalled `compatible_products` with its description and a few demonstrative examples in the prompt and found that REAPER is able to generate a valid plan. For the query *ink cartridge for HP 4650*, REAPER generated the plan in Figure 6. This also shows that REAPER has maintained its instruction following ability.

6 CONCLUSION

In this paper, we present REAPER – a reasoning-based planner – to generate retrieval plans for RAG dialog systems. The planner uses an instruction-tuned LLM and utilizes a novel data generation module to optimize the model for the retrieval planning while still retaining the ability to follow instructions in order to scale to new use-cases. Extensive experiments show that our model is 1) data-efficient – REAPER is trained on 6K in-domain queries, while classification models needed 150K and 2) easily scalable to new retrieval sources – we were able to add a new retrieval sources by increasing the training data by 286 in-domain queries. It also is an order of magnitude faster (207ms as compared to 2s) as compared to Agent based systems. Finally, we observe promising results indicating our model's capability to generalize to new tools and plan structures without explicitly being trained for it.

REFERENCES

[1] Anthropic. 2023. Model Card and Evaluations for Claude Models. (2023). https://cdn.sanity.io/files/4zrzovbb/website/bd2a28d2535bfb0494cc8e2a3bf135d2e7523226.pdf

[2] Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. RQ-RAG: Learning to Refine Queries for Retrieval Augmented Generation. *arXiv preprint arXiv:2404.00610* (2024).

[3] Christopher Clarke, Joseph Joshua Peper, Karthik Krishnamurthy, Walter Talamonti, Kevin Leach, Walter Lasecki, Yiping Kang, Lingjia Tang, and Jason Mars. 2022. One agent to rule them all: Towards multi-agent conversational AI. *arXiv preprint arXiv:2203.07665* (2022).

[4] Jiabao Fang, Shen Gao, Pengjie Ren, Xiuying Chen, Suzan Verberne, and Zhaochun Ren. 2024. A Multi-Agent Conversational Recommender System. *arXiv preprint arXiv:2402.01135* (2024).

[5] Ethan Fast, Binbin Chen, Julia Mendelsohn, Jonathan Bassen, and Michael S. Bernstein. 2018. Iris: A Conversational Agent for Complex Tasks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (, Montreal QC, Canada,) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3174047

[6] Yao Fu, Hao-Chun Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing Smaller Language Models towards Multi-Step Reasoning. *ArXiv* abs/2301.12726 (2023). https://api.semanticscholar.org/CorpusID:256390607

[7] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[8] Gautier Izacard and Edouard Grave. 2020. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *CoRR* abs/2007.01282 (2020). arXiv:2007.01282 https://arxiv.org/abs/2007.01282

[9] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Atlas: Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299* (2022).

[10] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity. *arXiv preprint arXiv:2403.14403* (2024).

[11] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[12] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. *CoRR* abs/2004.04906 (2020). arXiv:2004.04906 https://arxiv.org/abs/2004.04906

[13] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2022. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024* (2022).

[14] Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. 2022. Internet-augmented language models through few-shot prompting for open-domain question answering. *arXiv preprint arXiv:2203.05115* (2022).

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[16] Gabriel Resende Machado, Eugênio Silva, and Ronaldo Ribeiro Goldschmidt. 2021. Adversarial Machine Learning in Image Classification: A Survey Toward the Defender's Perspective. *ACM Comput. Surv.* 55, 1, Article 8 (nov 2021), 38 pages. https://doi.org/10.1145/3485133

[17] Yury A. Malkov and Dmitry A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR* abs/1603.09320 (2016). arXiv:1603.09320 http://arxiv.org/abs/1603.09320

[18] Vaibhav Mavi, Anubhav Jangra, and Adam Jatowt. 2022. A survey on multi-hop question answering and generation. *arXiv preprint arXiv:2204.09140* (2022).

[19] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019. Knowledge Guided Text Retrieval and Reading for Open Domain Question Answering. *CoRR* abs/1911.03868 (2019). arXiv:1911.03868 http://arxiv.org/abs/1911.03868

[20] Jeongeon Park, Bryan Min, Xiaojuan Ma, and Juho Kim. 2023. Choicemates: Supporting unfamiliar online decision-making with multi-agent conversational interactions. *arXiv preprint arXiv:2310.01331* (2023).

[21] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:258833055

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

[23] Wenhan Xiong, Xiang Lorraine Li, Srinivasan Iyer, Jingfei Du, Patrick S. H. Lewis, William Yang Wang, Yashar Mehdad, Wen-tau Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oguz. 2020. Answering Complex Open-Domain Questions with Multi-Hop Dense Retrieval. *CoRR* abs/2009.12756 (2020). arXiv:2009.12756 https://arxiv.org/abs/2009.12756

[24] Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323* (2023).

[25] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).

[26] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[27] Yue Yu, Chenyan Xiong, Si Sun, Chao Zhang, and Arnold Overwijk. 2022. COCO-DR: Combating Distribution Shifts in Zero-Shot Dense Retrieval with Contrastive and Distributionally Robust Learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 1462–1479.