InstructRAG: Leveraging Retrieval-Augmented Generation on Instruction Graphs for LLM-Based Task Planning

Zheng Wang*
Huawei Singapore Research Center
Singapore
wangzheng155@huawei.com

Shu Xian Teo*
Huawei Singapore Research Center
Singapore
teo.shu.xian@huawei.com

Jun Jie Chew
Huawei Singapore Research Center
Singapore
chew.jun.jie@huawei.com

Wei Shi
Huawei Singapore Research Center
Singapore
w.shi@huawei.com

Abstract

Recent advancements in large language models (LLMs) have enabled their use as agents for planning complex tasks. Existing methods typically rely on a thought-action-observation (TAO) process to enhance LLM performance, but these approaches are often constrained by the LLMs' limited knowledge of complex tasks. Retrieval-augmented generation (RAG) offers new opportunities by leveraging external databases to ground generation in retrieved information. In this paper, we identify two key challenges (enlargability and transferability) in applying RAG to task planning. We propose `InstructRAG`, a novel solution within a multi-agent meta-reinforcement learning framework, to address these challenges. `InstructRAG` includes a graph to organize past instruction paths (sequences of correct actions), an RL-Agent with Reinforcement Learning to expand graph coverage for enlargability, and an ML-Agent with Meta-Learning to improve task generalization for transferability. The two agents are trained end-to-end to optimize overall planning performance. Our experiments on four widely used task planning datasets demonstrate that `InstructRAG` significantly enhances performance and adapts efficiently to new tasks, achieving up to a 19.2% improvement over the best existing approach.

CCS Concepts

• **Information systems** → **Information retrieval**.

Keywords

large language model, retrieval-augmented generation, agent planning

ACM Reference Format:
Zheng Wang, Shu Xian Teo, Jun Jie Chew, and Wei Shi. 2025. InstructRAG: Leveraging Retrieval-Augmented Generation on Instruction Graphs for LLM-Based Task Planning. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25), July 13–18, 2025, Padua, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3726302.3730009

*Equal Contribution.

1 Introduction

With the significant advancement of large language models (LLMs), a recent trend has emerged in employing LLMs as intelligent agents to tackle diverse real-world planning tasks. These tasks include multi-hop reasoning [38], embodied tasks [24, 25, 34], web shopping [39], and scientific reasoning [29], etc. Many recent solutions to the planning problem, such as ReAct [41], KnowAgent [42], WKM [19], Reflexion [23], FireAct [3], NAT [30], and ETO [27], follow a thought-action-observation (TAO) process. In the thought phase, the LLM leverages its reasoning ability to create a plan by breaking down a task into a series of subtasks. In the action phase, the LLM determines the specific actions required, such as selecting which tool to use. In the observation phase, it captures the results of executing the action and provides feedback from the external environment to the LLM, facilitating the planning of subsequent TAO steps. Within this process, the thought and action are generated by the LLM, while the observation is implemented by the environment. Existing solutions adopt diverse strategies, including prompting [23, 41] or fine-tuning [3, 19, 27, 30, 42], to improve LLM-generated thoughts and actions for more effective planning. In particular, KnowAgent [42] integrates pre-defined rules into prompts to ensure that generated thoughts exhibit logical action transitions. For example, it prevents looking up an entity without first performing a search operation on the topic, as seen in HotPotQA [38]. Reflexion [23] incorporates self-reflection summaries into the TAO process to guide subsequent trials. WKM [19] trains a world knowledge model to generate thoughts based on knowledge acquired from human task-solving experiences.

While these existing methods aim to enhance LLM planning, they are often constrained by the inherent limitations of the LLMs themselves, such as their limited knowledge on complex tasks. The rapid development of retrieval-augmented generation (RAG) provides new opportunities to address these limitations by leveraging external databases. By anchoring LLM generation in retrieved information, RAG improves performance through the integration of relevant data during the planning process. In this context, we recognize that task-specific nature of information retrieval plays a crucial role for effective planning generation. For example, consider the question "Were Scott Derrickson and Ed Wood of the same nationality?" from HotPotQA, as shown in Figure 1(a). A potential retrieved plan for this question involves a sequence of actions (referred to as instruction paths in this paper): first, use Google Search to find information about Scott Derrickson (denoted by `Search[Scott Derrickson]`), then look up a sentence containing the keyword "nationality" (`Lookup[nationality]`), followed by `Search[Ed Wood]` and `Lookup[nationality]`. These instructions are specific to the question at hand and may vary depending on the topics or entities involved. A similar phenomenon can also be observed in ALFWorld embodied tasks, where instructions might include `Goto[shelf 6]`, then `Take[vase 2 from shelf 6]`.

In this paper, we discuss the task-specific nature in two aspects, with the goal of bridging the gap between task-specific questions and instructions derived from past experiences stored in a database through RAG. (1) Enlargeability: This refers to a task where the question falls *within* the scope of those covered by the external database. Specifically, we pre-store successful instruction paths in the database, with each path tailored to a specific task. To address questions related to these tasks, we explore a paradigm for combining instructions to expand the database's coverage. As illustrated in Figure 1(a), there are two successful instruction paths, P_1^1 and P_2^1, for solving questions Q_1^1 and Q_2^1, respectively. These paths consist of five instructions: searching for (a) Scott Derrickson, (b) Ed Wood, and (c) Christopher Nolan, and looking up entities related to (d) nationality and (e) birthplace. By combining these instructions in sequences such as $(a) \rightarrow (e) \rightarrow (c) \rightarrow (e)$, we can generate a new instruction path for solving a novel question (i.e., $\hat{Q}$) that was not covered by the original paths (but shares the same task of querying a location). The enlargeability is proposed to enhance the database's ability to address a wider range of questions. (2) Transferability: This refers to a task where the question is *outside* the scope of tasks covered by the external database. We note that LLM-based task planning is provided as a capability to support a wide range of tasks in practice. Transferability is essential for bridging the gaps between different tasks (i.e., those in the pre-built database and the questions at hand) within the RAG system. To achieve this, it is necessary to expand the database to incorporate new instructions required for different tasks, such as updating it with instructions relevant to the new tasks based on a development set. Additionally, certain trainable modules associated with the RAG can be rapidly adapted to accommodate the new tasks.

New Solution. Although recent research efforts [11, 12, 22] have attempted to apply RAG techniques to task planning, these methods often fall short in several aspects: i) [12] is primarily tailored for specific domains, such as decision-making in video games, making it challenging to generalize their designs to broader planning tasks as studied in this paper. ii) [22] focuses on multi-hop reasoning via search engines (e.g., using Google Search to access Wikipedia knowledge), but their effectiveness in tasks where the search engines are inapplicable (e.g., embodied tasks or web shopping) remains unexplored. iii) [11] simply relies on storing past experiences and retrieving similar ones using AKNN, without identifying key aspects such as enlargability and transferability. This gap results in suboptimal performance, as evidenced by our experiments.

To this end, we propose `InstructRAG`, a new solution based on a multi-agent meta-reinforcement learning framework. For (1), we design an instruction graph to instantiate the database. In this graph, nodes and edges represent two sets: nodes contain similar instructions, while edges represent corresponding tasks, all derived from successful instruction paths in past experiences. The rationale behind this approach is two-fold: 1) The graph provides a natural structure for organizing paths and facilitates the integration of new paths by clustering similar instructions related to various tasks. 2) Each node acts as a junction that enables the creation of new paths by combining stored instructions within it, and each edge records the tasks (with associated questions) along the path. This organization allows us to structure past experiences effectively within the database. Further, we design an RL-Agent that utilizes Reinforcement Learning to identify candidate paths on the graph, with the goal of optimizing the database's coverage to enhance its enlargeability. For (2), we explore a meta-learning approach into the RAG pipeline. Specifically, we introduce an additional agent, referred to as the ML-Agent, which Meta-Learns to select a path from the candidate paths provided by the RL-Agent. This selected path is then used as an in-context learning exemplar within the prompt, aiming to enhance the LLM's generalization to new tasks by updating it with only a few QA pairs during the meta-update phase. Here, the two agents collaborate within the TAO process [41] to facilitate task planning via grounding the generation of thoughts and actions by LLMs. We note that the RL-Agent generates candidate paths for the ML-Agent to select, and the ML-Agent then assesses the end-to-end effectiveness of the selected path, to incorporate this feedback as the reward for the RL-Agent. This interaction creates a positive loop, leading to improved planning performance.

To summarize, we make the following contributions.

- We conduct a systematic study of leveraging RAG for LLM-based task planning, and identify two key properties (i.e., enlargability and transferability) that a potential technique should possess. To our best knowledge, this is the first attempt of its kind.
- We propose a new solution called `InstructRAG`, which includes three key components: an instruction graph, an RL-Agent, and an ML-Agent. These components are integrated into a multi-agent meta-reinforcement learning framework that explicitly trains to optimize end-to-end task planning performance.
- We conduct extensive experiments on four widely used task planning datasets: HotpotQA [38], ALFWorld [25], Webshop [39], and ScienceWorld [29], across three typical LLMs. Our `InstructRAG` can be integrated with both trainable LLMs (e.g., GLM-4 [9]) for fine-tuning and frozen LLMs (e.g., GPT-4o mini [1] and DeepSeek-V2 [7]). The results demonstrate that `InstructRAG` improves performance by approximately 19.2%, 9.3%, 6.1%, and 10.2% over the best baseline method on the four datasets, respectively. In addition, `InstructRAG` adapts rapidly to new tasks, achieving effective performance with few-shot learning.

2 Related Work

LLM-based Agent Planning. To solve complex tasks, humans typically decompose them into smaller sub-tasks and then evaluate the plan's effectiveness. Similarly, LLM-based agents follow this routine, and we categorize existing techniques based on whether the agent receives feedback during the planning process. A detailed survey of LLM-based agent planning can be found in [28, 37]. *In* *planning without feedback*, agents do not receive feedback that influences their future actions. The main techniques for this category include (1) single-path reasoning [21, 35], (2) multi-path reasoning [2, 31, 40], and (3) using an external planner [5, 13]. Specifically, for (1), CoT [35] illustrates the reasoning steps for LLMs to tackle complex tasks using prompts, thereby guiding LLMs to plan and execute actions step-by-step. For (2), CoT-SC [31] explores diverse reasoning paths to solve complex tasks. Initially, it utilizes CoT to generate multiple reasoning paths and their respective answers. Subsequently, it selects the answer with the highest frequency as the final output. For (3), external planners are designed to generate plans for specific domains. For example, LLM+P [13] focuses on robot planning tasks by defining them using formal Planning Domain Definition Languages (PDDL). It utilizes an external planner, such as the Fast Downward planner [10], which employs heuristic search to handle PDDL formulations. The results generated by the planner are then translated back into natural language by LLMs.

In planning with feedback, effectiveness is generally improved by receiving feedback after actions are taken, which supports long-horizon planning. This feedback can come from (1) environments [3, 41], (2) humans [19, 42], and (3) models [16, 23]. For (1), ReAct [41] proposes the TAO process, where a language model generates the thought for planning, the action involves interacting with the environment, and the observation consists of external feedback (such as search engine results) based on the action. FireAct [3] generates the TAO using various methods, which are then converted into the ReAct format to fine-tune a small language model. For (2), KnowAgent [42] integrates action knowledge, which includes rules determining action transitions, into prompts to enhance the planning capabilities of LLMs. This knowledge is derived from both human input and GPT-4 [1]. Further, WKM [19] is introduced to facilitate agent planning using a world knowledge model. This model is trained by comparing selected trajectories (annotated by humans) with rejected trajectories (explored by an experienced agent). For (3), Reflexion [23] employs verbal feedback to enhance the agent's planning based on previous failures. It transforms binary or scalar feedback from self-evaluation into a textual summary, which is then added as additional context for the agent in subsequent planning. In this paper, we explore a new RAG-based approach to task planning, emphasizing two key properties: enlargability and transferability in technique development.

Retrieval-Augmented Generation. RAG enhances LLM generation by querying an external database to obtain relevant information, which grounds the subsequent text generation. Recent studies utilize RAG for task planning [11, 12, 22, 33]. Specifically, RAT [33] enhances CoT by iteratively revising each thought step with retrieved information relevant to the task query, thereby improving LLMs' ability to reason over long-horizon generation tasks. RAP [11] stores past experiences, including context and action-observation trajectories, and retrieves them based on their similarity to the current situation. The goal is to facilitate deriving appropriate actions by leveraging memory examples from similar tasks. PlanRAG [12] is designed for decision QA tasks, following a plan-then-retrieval approach. The LLM first generates a plan to guide the analysis, then retrieves information from an external database by formulating queries. It also continuously evaluates the need for re-planning during the process. GenGround [22] explores a generate-then-ground approach for multi-hop reasoning tasks. It breaks down a complex question into sub-questions, generates an immediate answer for each, then revises it with retrieved information. This revised answer informs the next sub-question, iterating until the final answer is achieved. In this paper, we propose `InstructRAG` within a multi-agent meta-reinforcement learning framework to systematically address the gap in leveraging RAG for task-specific questions and stored past experiences.

Meta-learning for Improving LLMs via In-context Learning (ICL). To enhance the transferability of LLMs to unseen tasks, meta-learning approaches [4, 6, 18, 26] have been developed. These approaches fine-tune pre-trained LLMs using a diverse set of tasks, formatted as ICL instances by pre-appending task-specific exemplars to the prompts during training. These methods follow Model-Agnostic Meta-Learning (MAML) principles [8]. Specifically, MAML-en-LLM [26] explores a wide parameter space to learn truly generalizable parameters that perform well on disjoint tasks and adapt effectively to unseen tasks. MTIL [6] investigates the application of meta-learning to multi-task instructional learning [32], aiming to enhance generalization to unseen tasks in a zero-shot setting. MetaICL [18] adapts a LLM to perform in-context learning across a broad range of training tasks. It aims to improve the model's ability to learn new tasks in context during testing, by conditioning on a few training examples without requiring parameter updates or task-specific templates. In this paper, we propose a novel meta-reinforcement learning framework to improve transferability, with two cooperative agents tailored for planning tasks. This approach is distinctly different from existing methodologies in the field.

3 Problem Statement

We explore the problem of LLM-based task planning through RAG, grounded in an external database (i.e., instruction graph). In this context, we identify two practical properties that should be met:

- Enlargeability: It should expand the scope of the instruction graph by traversing existing instructions (nodes) on the graph and combining them into new sequences of instructions (paths). This will help the LLM in completing tasks that do not have pre-defined paths during the graph's construction.
- Transferability: Task planning as a capability in practice involves developing techniques that achieve rapid adaptation to new tasks. For example, the trained model should be able to quickly learn a new task from a small amount of new data.

4 Methodology

4.1 Overview of `InstructRAG`

The proposed `InstructRAG` tackles the challenges of LLM-based task planning by focusing on two key properties: enlargeability and transferability. It comprises several components: instruction graph construction (Section 4.2), RL-Agent (Section 4.3), and ML-Agent (Section 4.4). These components are integrated into a multi-agent meta-reinforcement learning framework, which is detailed in terms of three stages: training, few-shot learning, and testing (Section 4.5).

Training. We illustrate the overall framework in Figure 1. Specifically, the training tasks (seen tasks) are divided into a support

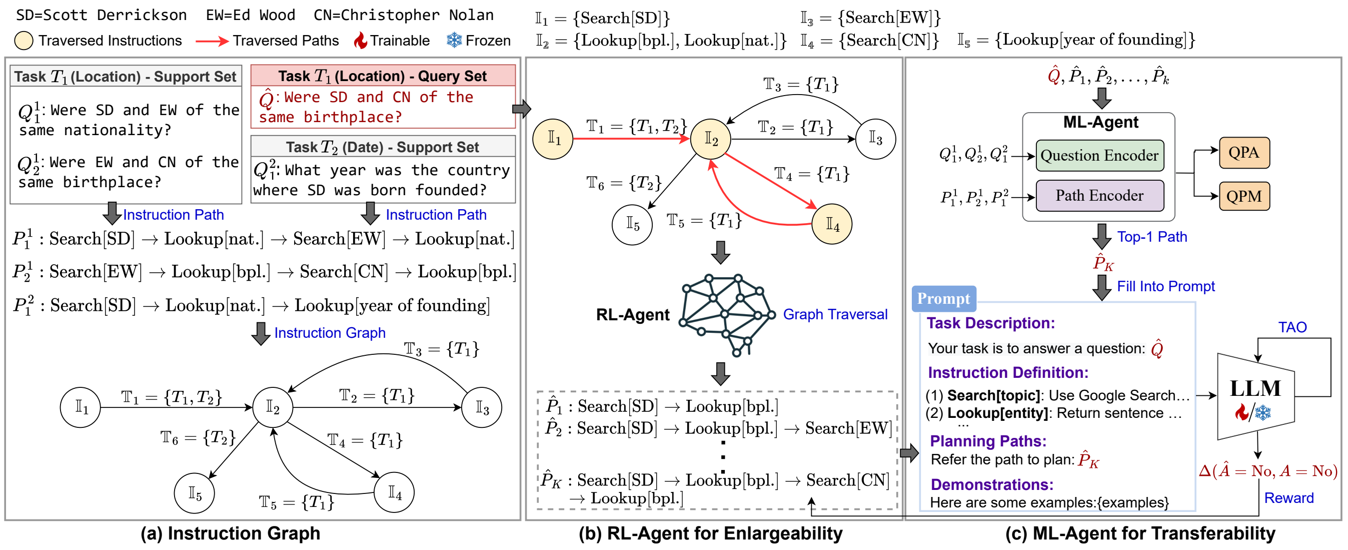

SD=Scott Derrickson EW=Ed Wood CN=Christopher Nolan
Traversed Instructions
Traversed Paths
Trainable
Frozen
$\mathbb{I}_1 = \{\text{Search[SD]}\}$
$\mathbb{I}_2 = \{\text{Lookup[bpl.], Lookup[nat.]}\}$
$\mathbb{I}_3 = \{\text{Search[EW]}\}$
$\mathbb{I}_4 = \{\text{Search[CN]}\}$
$\mathbb{I}_5 = \{\text{Lookup[year of founding]}\}$
Task T_1 (Location) - Support Set
Q_1^1: Were SD and EW of the same nationality?
Q_2^1: Were EW and CN of the same birthplace?
Task T_1 (Location) - Query Set
$\hat{Q}$: Were SD and CN of the same birthplace?
Task T_2 (Date) - Support Set
Q_1^2: What year was the country where SD was born founded?
Instruction Path
Instruction Path
P_1^1 : Search[SD] → Lookup[nat.] → Search[EW] → Lookup[nat.]
P_2^1 : Search[EW] → Lookup[bpl.] → Search[CN] → Lookup[bpl.]
P_1^2 : Search[SD] → Lookup[nat.] → Lookup[year of founding]
Instruction Graph
$\mathbb{T}_1 = \{T_1, T_2\}$
$\mathbb{T}_2 = \{T_1\}$
$\mathbb{T}_3 = \{T_1\}$
$\mathbb{T}_4 = \{T_1\}$
$\mathbb{T}_5 = \{T_1\}$
$\mathbb{T}_6 = \{T_2\}$
RL-Agent
Graph Traversal
$\hat{P}_1$: Search[SD] → Lookup[bpl.]
$\hat{P}_2$: Search[SD] → Lookup[bpl.] → Search[EW]
$\hat{P}_K$: Search[SD] → Lookup[bpl.] → Search[CN] → Lookup[bpl.]
$\hat{Q}, \hat{P}_1, \hat{P}_2, \ldots, \hat{P}_k$
ML-Agent
Q_1^1, Q_2^1, Q_1^2
P_1^1, P_2^1, P_1^2
Question Encoder
Path Encoder
QPA
QPM
Top-1 Path
$\hat{P}_K$
Fill Into Prompt
Prompt
Task Description:
Your task is to answer a question: $\hat{Q}$
Instruction Definition:
(1) Search[topic]: Use Google Search…
(2) Lookup[entity]: Return sentence …
Planning Paths:
Refer the path to plan: $\hat{P}_K$
Demonstrations:
Here are some examples:{examples}
TAO
LLM
$\Delta(\hat{A} = \text{No}, A = \text{No})$
Reward
(a) Instruction Graph
(b) RL-Agent for Enlargeability
(c) ML-Agent for Transferability

Figure 1: Architecture of the proposed InstructRAG with multi-agent meta-reinforcement learning, illustrated on HotpotQA.

set and a query set. For support set, it is used to construct the instruction graph by extracting instruction paths from questions and forming the graph based on these paths. Additionally, the paths and corresponding questions help warm-start the RL-Agent, and pre-train the ML-Agent with two pre-training tasks: Question Path Alignment (QPA) and Question Path Matching (QPM). For query set, it is used to query the graph and trains the RL-Agent and the ML-Agent within a multi-agent framework. The RL-Agent finds candidate paths through graph traversal, which is modeled as a Markov Decision Process (MDP) and optimized using reinforcement learning. The RL-Agent is trained to handle questions not seen during the graph construction, enlarging the capability of the instruction graph by combining instructions into the paths to address these questions. The ML-Agent then selects the most relevant path among the candidate paths based on their representations, which is used to form the prompt for a LLM to predict the final answer, following the TAO process. We note that the transferability is considered through an in-context learning manner, where the LLM learns a new task by conditioning on the task-specific path in the prompt. The ML-Agent optimizes this process, either with a trainable or frozen LLM, via meta-learning.

Few-shot Learning and Testing. Once the parameters of the RL-Agent and ML-Agent are meta-trained, we rapidly adapt the model parameters using few-shot examples from the support set on testing tasks (unseen tasks). The testing is then conducted based on the query set of these testing tasks.

4.2 Instruction Graph

Instruction. An instruction I represents a specific action performed by LLMs, e.g., `Search[Scott Derrickson]` is an instruction, meaning to use Google Search to find relevant information about Scott Derrickson as shown in Figure 1(a).

Instruction Path. An instruction path $P_i^j = \langle I_1, I_2, \ldots, I_{|P|} \rangle$ is represented as a sequence of instructions that LLMs follow step-by-step to perform actions and complete the i-th question of the j-th task, e.g., P_1^2: `Search[Scott Derrickson]` $\rightarrow$ `Lookup[nationality]` $\rightarrow$ `Lookup[year of founding]` denotes an instruction path to address the question Q_1^2 of the task T_2 as shown in Figure 1(a).

Instruction Graph. An instruction graph $G(\mathbb{V}, \mathbb{E})$ is represented as a directed graph that organizes instruction paths of questions belonging to various tasks, where $\mathbb{V}$ and $\mathbb{E}$ represent the nodes and edges of the graph, respectively. Each node $\mathbb{I} \in \mathbb{V}$ denotes an instruction set, i.e., $\mathbb{I} = \{I_1, I_2, \ldots, I_{|\mathbb{I}|}\}$, clustering similar instructions. Each edge $\mathbb{T} \in \mathbb{E}$ denotes a task set, i.e., $\mathbb{T} = \{T_1, T_2, \ldots, T_{|\mathbb{T}|}\}$, recording the tasks with associated questions involved on the path.

The Graph Construction and Insights. We present the graph construction in two steps, illustrated with a running example in Figure 1(a). The detailed process is outlined in Algorithm 1.

Step-1 (Generating Instruction Paths): We divide the dataset into two parts: the support set and the query set, following the meta-learning setup. The support set is used for graph construction, while the query set is used to query the graph to train enlargeability and transferability, to be discussed in Section 4.3 and Section 4.4, respectively. For each question in the support set, we generate its instruction path using existing task planning techniques [23, 41, 42]. We select the path that correctly plans the question for construction, ensuring the planning is grounded in the prepared database, aligning with the goal of RAG.

Step-2 (Inserting Instructions with a Threshold δ): Then, we iteratively insert each instruction in the generated paths, i.e., P_1^1, P_2^1, P_1^2, into the graph G. The first two instructions in P_1^1 are initialized to create two node sets, i.e., $\mathbb{I}_1 \longleftarrow$ `Search[Scott Derrickson]` and $\mathbb{I}_2 \longleftarrow$ `Lookup[nationality]`, which correspond to an edge set $\mathbb{T}_1$ recording the involved task $\{T_1\}$. Here, we note that adjacent instructions are not inserted into the same node, as this would break the transition between them. To insert the next instruction `Search[Ed Wood]`, we perform an AKNN search [17] on all instructions excluding the instructions in its adjacent node (i.e., `Lookup[nationality]` $\in \mathbb{I}_2$), identifying the most similar instruction `Search[Scott Derrickson]`, which is associated with a similarity value (e.g., cosine similarity) ψ in the node set $\mathbb{I}_1$. Then, we define a threshold δ to control the insertion. If $\psi < \delta$, a new node set $\mathbb{I}_3$ is created and the instruction is inserted into this new node, i.e., $\mathbb{I}_3 \leftarrow$ `Search[Ed Wood]`; otherwise, the instruction is inserted into the identified node $\mathbb{I}_1$. The process continues until all instructions are inserted. Additionally, we note that when the instruction `Lookup[nationality]` of P_1^2 is inserted into $\mathbb{I}_2$ (with a cosine similarity of 1.0), the task T_2 is also added to the edge set $\mathbb{T}_1$, resulting in $\{T_1, T_2\}$.

We present two key insights into graph construction: (1) Graphs naturally organize instruction paths, where nodes and edges are represented as sets to enable flexible integration of similar instructions across tasks. (2) The threshold controls instruction similarity, forming junction nodes that create new paths beyond those in the original data. For instance, merging `Lookup[nationality]` and `Lookup[birthplace]` into $\mathbb{I}_2$ enables novel paths like $\mathbb{I}_1 \rightarrow \mathbb{I}_2 \rightarrow \mathbb{I}_4 \rightarrow \mathbb{I}_2$, thus improving graph expandability to cover more questions (e.g., $\hat{Q}$ in Figure 1(a)).

Algorithm 1: The Instruction Graph Construction

```
Require : a support set $\mathbb{S}$; a threshold $\delta$
1  $IC \leftarrow 3, TC \leftarrow 2$ // two counters for node and edge sets
2  for each $T_j \in \mathbb{S} (1 \leq j \leq |\mathbb{S}|)$ do
3      for each $Q_i^j \in T_j (1 \leq i \leq |T_j|)$ do
4          obtain a correct $P_i^j = \langle I_1, I_2, \ldots, I_{|P_i^j|} \rangle$ for $Q_i^j$
5          $\mathbb{I}' \leftarrow \emptyset$ // record the last node set
6          for $k = 1, 2, ..., |P_i^j|$ do
7              if $i = 1$ and $j = 1$ and $k < 3$ then
8                  $\mathbb{I}_1.\text{add}(I_1), \mathbb{I}_2.\text{add}(I_2), \mathbb{T}_1 \leftarrow \text{Edge}(\mathbb{I}_1, \mathbb{I}_2)$
9                  $\mathbb{T}_1.\text{add}(T_1), G.\text{addEdge}(\mathbb{T}_1), \mathbb{I}' \leftarrow \mathbb{I}_2$
10                 continue
11             recall $\mathbb{I}_s$ and $\psi$ for $I_k$ with AKNN on $G.\mathbb{V} - \mathbb{I}'$
12             if $\psi < \delta$ then
13                 $\mathbb{I}_{IC}.\text{add}(I_k), \mathbb{T}_{TC} \leftarrow \text{Edge}(\mathbb{I}', \mathbb{I}_{IC})$
14                 $\mathbb{T}_{TC}.\text{add}(T_j)$ , $G.\text{addEdge}(\mathbb{T}_{TC})$
15                 $\mathbb{I}' \leftarrow \mathbb{I}_{IC}, IC \leftarrow IC + 1, TC \leftarrow TC + 1$
16             else
17                 $\mathbb{I}_s.\text{add}(I_k)$
18                 if $Edge(\mathbb{I}', \mathbb{I}_s) \in G$ then
19                     obtain the edge of $(\mathbb{I}', \mathbb{I}_s)$ denoted by $\mathbb{T}'$
20                     $\mathbb{T}'.\text{add}(T_j)$
21                 else
22                     $\mathbb{T}_{TC} \leftarrow \text{Edge}(\mathbb{I}', \mathbb{I}_s), \mathbb{T}_{TC}.\text{add}(T_j)$
23                     $G.\text{addEdge}(\mathbb{T}_{TC}), TC \leftarrow TC + 1$
24                 $\mathbb{I}' \leftarrow \mathbb{I}_s$
25 Return the instruction graph $G$
```

4.3 RL-Agent: Retrieving Instruction Paths on Instruction Graph

Given an instruction graph G, we explore its enlargeability through graph traversal to retrieve various instruction paths that solve questions denoted by $\hat{Q}$ not present during construction (i.e., questions in the query set). To achieve this, we train an agent for traversal, which examines each path in the graph, e.g., via depth-first search (DFS). For each node, the agent decides whether to include or exclude the node (i.e., actions) in the path based on the instructions contained in the node and the tasks connected by its edges (i.e., states). A high-quality retrieved path benefits subsequent planning, reflected by an end-to-end metric such as F1 scores on HotPotQA [38] (i.e., rewards), which can then inform instruction selection. This process forms a Markov Decision Process (MDP), and we employ Reinforcement Learning (RL) to optimize it.

Constructing Decision Environment. The instruction graph G typically contains numerous instruction paths, formed by combining different instructions at each node. To manage this, we limit the RL-Agent's retrieval to K relevant instruction paths, denoted as $\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_K$, which are then utilized for planning in the next phase by the ML-Agent (to be introduced in Section 4.4), where the K is a hyperparameter that can be tuned for optimal performance. We first perform an AKNN search on all instructions for a query $\hat{Q}$. The agent's traversal starts from the most similar instructions (corresponding to the nodes) using DFS. Once the agent excludes a node and backtracks to another branch, an instruction path is formed. This process continues until K paths are retrieved.

States. Suppose we have an input question $\hat{Q}$ and visit a node $\mathbb{I}$ (a set of instructions), along with its in-degree edge $\mathbb{T}$ (a set of tasks). We define the state $\mathbf{s}$ using three cosine similarities $CS(\cdot, \cdot)$, that is

$$\mathbf{s} = \{\max_{I_i \in \mathbb{I}} CS(\mathbf{v}_{\hat{Q}}, \mathbf{v}_{I_i}), \max_{T_j \in \mathbb{T}} CS(\mathbf{v}_{\hat{Q}}, \mathbf{v}_{T_j}), \max_{Q_k^c \in T_c} CS(\mathbf{v}_{\hat{Q}}, \mathbf{v}_{Q_k^c})\},$$
$$\mathbf{v}_{T_j} = \frac{1}{|T_j|} \sum_{k=1}^{|T_j|} \mathbf{v}_{Q_k^j} \text{ and } c = \arg\max_{T_c \in \mathbb{T}} CS(\mathbf{v}_{\hat{Q}}, \mathbf{v}_{T_c}), \quad (1)$$

where $\mathbf{v}_{\cdot}$ denotes an embedding vector. We construct the state by (1) examining the most similar instruction in the node, (2) identifying the most similar task, denoted by T_c, in the edge (whose embedding is calculated as the average of the question embeddings belonging to the task), and (3) finding the most similar question within T_c.

Actions. Let a denote an action, which has two choices during the graph traversal: including the visited node by selecting the most similar instruction into the path $\hat{P}_i$ $(1 \leq i \leq K)$ and searching its connected nodes, or excluding the node and backtracking the search from another branch, then an instruction path is formed. The action a is formally defined as:

$$a = 1 \text{ (including) or } 0 \text{ (excluding)}. \quad (2)$$

Considering the consequence of performing an action, it transitions the environment to the next state $\mathbf{s}'$, affecting which node or edge is used for constructing the state. Notably, some predefined rules may be further incorporated to constrain the action space (e.g., a rule of avoiding `Lookup` information without first performing `Search` on HotpotQA [42]), which benefits more accurate path selection.

Rewards. Let r denote a reward, which corresponds to the end-to-end feedback of an instruction path that contributes to the generated answer $\hat{A}$ by a LLM for $\hat{Q}$. Specifically, when an instruction path from the K paths is selected and written into the prompt by the ML-Agent, the LLM generates an answer $\hat{A}$. This answer can be evaluated using a specific metric $\Delta(\cdot, \cdot)$ (e.g., F1 score), defined as:

$$r = \Delta(\hat{A}, A), \quad (3)$$

where A denotes the ground truth answer. The rationale for designing the reward is to enable joint optimization for the two agents in a multi-agent setup, where the RL-Agent provides paths for the ML-Agent to write into the prompt, and the feedback from the prompt affects the path retrieval by the RL-Agent. Therefore, the two agents can be jointly optimized to improve the overall performance.

Policy Learning. We involve two phases for training the MDP policy: warm-start (WS) and policy gradient (PG). In WS, the goal is to equip the agent with the basic ability to include or exclude instructions. To achieve this, we randomly sample questions from the support set. For each question, we randomly sample nodes on G and construct its state using Equation 1. If the node is on the instruction path for the question, the state is associated with an action labeled 1; otherwise, it is labeled 0. We collect these state-action pairs and train the RL-Agent using binary cross-entropy:

$$\mathcal{L}_{\mathrm{WS}} = -y * \log(P) + (y-1) * \log(1-P), \tag{4}$$

where y denotes the label, and P is the predicted probability of the positive class. In PG, the primary goal is to develop a policy $\pi_\theta(a|\mathbf{s})$ that guides the agent in performing actions a based on the given states $\mathbf{s}$ for questions on the query set, with the aim of maximizing the cumulative reward R. We employ the REINFORCE algorithm [36] to learn this policy, where θ represents the parameters of the RL-Agent. The loss function is defined as:

$$\mathcal{L}_{\mathrm{PG}} = -R \ln \pi_\theta(a|\mathbf{s}). \tag{5}$$

4.4 ML-Agent: Generating Prompts for Planning

In the ML-Agent, the most relevant path identified by the RL-Agent is selected and integrated into the prompt for a LLM. We manage transferability through the ML-Agent using Meta-Learning (ML). The rationale is that the agent is trained to structure the prompt as an in-context learning (ICL) instance by pre-appending the exemplar planning path, which can potentially improve LLM generalization to new tasks by updating with only a few examples, as evidenced in [18, 26]. Below, we discuss the model architecture and training details for the ML-Agent.

Model Architecture. As shown in Figure 1(c), our ML-Agent uses the text encoder structure from [20] for both the question encoder and the path encoder. It employs two transformer modules with shared self-attention layers to capture potential features. We treat the instruction path and question as two text sequences ending with `[EOS]` tokens, and derive their feature representations from the activations of the highest transformer layer at these `[EOS]` tokens. The ML-Agent is trained to align the question and instruction path representations, and the most relevant path is retrieved based on the cosine similarities of these representations. Notably, the model does not use a K-classifier for path selection, ensuring that the architecture remains independent of the K hyperparameter and does not require retraining when K is adjusted.

Training ML-Agent. The ML-Agent training involves two phases: pre-training (PT) and fine-tuning (FT). In PT, we optimize the agent using two pre-training tasks: Question Path Alignment (QPA) and Question Path Matching (QPM). For QPA, the objective is to align question and path representations by bringing similar pairs closer together and pushing dissimilar pairs apart through a contrastive approach. Specifically, we sample a batch of question-path pairs from the support set (e.g., Q_1^1 and P_1^1 as shown in Figure 1). For each pair, denoted by $< Q_i^j, P_i^j >$, where $Q_i^j \in \mathcal{Q}$ and $P_i^j \in \mathcal{P}$, we obtain their embedding vectors $\mathbf{v}_{i,j}^Q$ and $\mathbf{v}_{i,j}^P$ via the two encoders. We treat $\mathbf{v}_{i,j}^P$ as the positive example for $\mathbf{v}_{i,j}^Q$ (the anchor), since Q_i^j and P_i^j are paired, while the other paths in the batch are considered as negatives. The contrastive loss, denoted as $\mathcal{L}_{Q,P}$, encourages the paths to align with the anchor question by comparing their positive and negative pairs, that is

$$\mathcal{L}_{Q,P} = \sum\nolimits_{Q_i^j \in \mathcal{Q}} -\log \frac{\exp(\mathbf{v}_{i,j}^Q \cdot \mathbf{v}_{i,j}^P / \tau)}{\sum_{P_{i'}^{j'} \in \mathcal{P}, i' \neq i, j' \neq j} \exp(\mathbf{v}_{i,j}^Q \cdot \mathbf{v}_{i',j'}^P / \tau)}, \tag{6}$$

where τ denotes a temperature parameter. Symmetrically, we can define $\mathcal{L}_{P,Q}$ by anchoring at $\mathbf{v}_{i,j}^P$. The overall loss $\mathcal{L}_{\mathrm{QPA}}$ is then defined as:

$$\mathcal{L}_{\mathrm{QPA}} = (\mathcal{L}_{Q,P} + \mathcal{L}_{P,Q})/2. \tag{7}$$

For QPM, we align questions with paths through a binary classification task. The model predicts whether a question-path pair is a match (labeled 1) or a mismatch (labeled 0). The training objective uses binary cross-entropy loss, which is defined as follows:

$$\mathcal{L}_{\mathrm{QPM}} = -y * \log(P) + (y-1) * \log(1-P). \tag{8}$$

where y denotes the label and P represents the predicted probability of the positive class. Finally, the ML-Agent is trained using a multi-task learning approach, with the loss function $\mathcal{L}_{\mathrm{PT}}$ is defined as:

$$\mathcal{L}_{\mathrm{PT}} = \mathcal{L}_{\mathrm{QPA}} + \mathcal{L}_{\mathrm{QPM}}. \tag{9}$$

In FT, we further fine-tune the model using questions from the query set. Specifically, for each question $\hat{Q} \in \hat{\mathcal{Q}}$, we retrieve K paths using the RL-Agent. We employ a hard negative mining strategy where the K retrieved paths are considered as hard negative samples, since they are relevant to the question $\hat{Q}$. Additionally, we sample paths from other questions and add them to the K paths, forming a path pool denoted by $\hat{\mathcal{P}}$. The performance is then evaluated by comparing the ground truth answer A with the generated answer $\hat{A}$ via a LLM for each path in $\hat{\mathcal{P}}$. Based on a specific metric $\Delta(\hat{A}, A)$, the best path denoted by $\hat{P}$ is identified as the positive example for $\hat{Q}$, and the other paths in the pool are considered as negatives. The loss function $\mathcal{L}_{\mathrm{FT}}$ for the fine-tuning phase is defined as:

$$\begin{aligned} \mathcal{L}_{\mathrm{FT}} &= (\mathcal{L}'_{Q,P} + \mathcal{L}'_{P,Q})/2 \\ \mathcal{L}'_{Q,P} &= \sum\nolimits_{\hat{Q} \in \hat{\mathcal{Q}}} -\log \frac{\exp\left(\mathbf{v}^{\hat{Q}} \cdot \mathbf{v}^{\hat{P}} / \tau\right)}{\sum_{\overline{P} \in \hat{\mathcal{P}}, \overline{P} \neq \hat{P}} \exp\left(\mathbf{v}^{\hat{Q}} \cdot \mathbf{v}^{\overline{P}} / \tau\right)}, \end{aligned} \tag{10}$$

where $\mathbf{v}^{\hat{Q}}$ and $\mathbf{v}^{\hat{P}}$ denote the embedding vectors for $\hat{Q}$ and $\hat{P}$, respectively. $\mathcal{L}'_{P,Q}$ is a symmetric definition based on $\mathcal{L}'_{Q,P}$.

Prompt Structure for LLM Generation. The path $\hat{P}$ returned by ML-Agent is used to construct a prompt that guides the LLM in generating an answer, denoted as $\hat{A}$. Our prompt is composed of four parts, as illustrated in Figure 1(c). (1) Task Description: This part introduces the task, detailing the specific question $\hat{Q}$ to be solved. (2) Instruction Definitions: This part provides definitions for each instruction, such as `Search[topic]` or `Lookup[entity]`. (3) Planning Path: The path $\hat{P}$ is integrated to create a structured plan, guiding the LLM through step-by-step actions to address $\hat{Q}$. (4) Demonstrations: Examples of planning paths are provided to offer reference and context for the LLM. Additionally, the `InstructRAG` framework supports integration with both trainable LLMs (e.g., fine-tuning GLM-4 [9] with the ground truth paths following [42]), and frozen LLMs (e.g., GPT-4o mini [1] and DeepSeek-V2 [7]) to leverage its inherent capabilities for planning.

Algorithm 2: The `InstructRAG` - Training Stage

```
Require :a training support set $\mathbb{S}$; a training query set $\mathbb{Q}$
1  randomly initialize $\theta$ for RL-Agent and $\eta$ for ML-Agent
2  construct the instruction graph $G$ with $\mathbb{S}$ by Algorithm 1
3  while not done do
4      sample a batch of tasks $\mathcal{T}$
5      for each $T_i \in \mathcal{T} (1 \leq i \leq |\mathcal{T}|)$ do
6          evaluate $\nabla_\theta \mathcal{L}_{\mathrm{WS}}^{T_i}(\text{RL-Agent}_\theta)$ by Eq 4 wrt $\mathcal{B}$ questions for
             $T_i$ in $\mathbb{S}$
7          compute adapted $\theta'_i \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\mathrm{WS}}^{T_i}(\text{RL-Agent}_\theta)$
8          evaluate $\nabla_\theta \mathcal{L}_{\mathrm{PT}}^{T_i}(\text{ML-Agent}_\eta)$ by Eq 9 wrt $\mathcal{B}$ questions for
             $T_i$ in $\mathbb{S}$
9          compute adapted $\eta'_i \leftarrow \eta - \alpha \nabla_\theta \mathcal{L}_{\mathrm{PT}}^{T_i}(\text{ML-Agent}_\eta)$
10     update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i} \mathcal{L}_{\mathrm{PG}}^{T_i}(\text{RL-Agent}_{\theta'_i})$ by Eq 5 wrt
         questions for all sampled tasks in $\mathbb{Q}$
11     update $\eta \leftarrow \eta - \beta \nabla_\eta \sum_{T_i} \mathcal{L}_{\mathrm{FT}}^{T_i}(\text{ML-Agent}_{\eta'_i})$ by Eq 10 wrt
         questions for all sampled tasks in $\mathbb{Q}$
12 Return trained RL-Agent$_\theta$ and ML-Agent$_\eta$
```

Algorithm 3: The `InstructRAG` - Few-Shot Learning Stage

```
Require :a testing support set $\mathbb{S}'$; RL-Agent$_\theta$; ML-Agent$_\eta$
1  insert $\mathbb{S}'$ into $G$ by Algorithm 1, and obtain $G'$
2  for each $T_i \in \mathbb{S}' (1 \leq i \leq |\mathbb{S}'|)$ do
3      $\theta'_i \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\mathrm{WS}}^{T_i}(\text{RL-Agent}_\theta) - \beta \nabla_\theta \mathcal{L}_{\mathrm{PG}}^{T_i}(\text{RL-Agent}_\theta)$ by
         Eq 4 and Eq 5 wrt $\mathcal{B}$ questions for $T_i$ in $\mathbb{S}'$
4      $\eta'_i \leftarrow \eta - \alpha \nabla_\eta \mathcal{L}_{\mathrm{PT}}^{T_i}(\text{ML-Agent}_\eta) - \beta \nabla_\eta \mathcal{L}_{\mathrm{FT}}^{T_i}(\text{ML-Agent}_\eta)$ by
         Eq 9 and Eq 10 wrt $\mathcal{B}$ questions for $T_i$ in $\mathbb{S}'$
5  Return adapted RL-Agent$_{\theta'_i}$ and ML-Agent$_{\eta'_i}$ for each task
```

Algorithm 4: The `InstructRAG` - Testing Stage

```
Require :a testing query set $\mathbb{Q}'$; RL-Agent$_{\theta'_i}$; ML-Agent$_{\eta'_i}$
1  for each $T_i \in \mathbb{Q}' (1 \leq i \leq |\mathbb{Q}'|)$ do
2      run RL-Agent$_{\theta'_i}$ and ML-Agent$_{\eta'_i}$ for questions in $T_i$
3      evaluate the effectiveness with a metric $\Delta(\cdot, \cdot)$
4  Return the average effectiveness across all tasks
```

4.5 The `InstructRAG` Framework

We present the `InstructRAG` framework in three stages: (1) the Training Stage, (2) the Few-Shot Learning Stage, and (3) the Testing Stage. In (1), the framework employs a meta-learning approach [8] to collaboratively train two agents using both support and query sets from seen tasks. In (2), the agents' parameters are quickly adapted to unseen tasks using few-shot examples on the support set. In (3), the effectiveness of the adaptation is evaluated using the query set on these unseen tasks.

Training Stage. As shown in Algorithm 2, the process inputs a support set and a query set from the seen training tasks and outputs the trained RL-Agent and ML-Agent. The support set is used to construct the instruction graph G as detailed in Algorithm 1. The two agents are then trained iteratively. In each iteration, the RL-Agent and ML-Agent are represented as RL-Agent$_\theta$ and ML-Agent$_\eta$ with parameters θ and η, respectively. When adapting to a new task T_i, the parameters θ and η are updated to θ' and η' using Equations 4 and 9 based on the support set (α denotes a learning rate). The updated parameters are quickly computed using one or more gradient descent updates with $\mathcal{B}$ questions. Following this, the model parameters are optimized by improving the performance of RL-Agent$_{\theta'_i}$ using Equation 5 and ML-Agent$_{\eta'_i}$ using Equation 10, with respect to θ and η across sampled tasks from the query set (β denotes a learning rate). Our training approach aims to optimize both agents so that a minimal number of gradient steps on a new task will produce the most effective behavior for that task.

Few-shot Learning Stage. As shown in Algorithm 3, the process adapts the trained RL-Agent$_\theta$ and ML-Agent$_\eta$ to separate models, denoted as RL-Agent$_{\theta'_i}$ and ML-Agent$_{\eta'_i}$, for each task T_i. This adaptation involves extending the graph G to G' using Algorithm 1 on the testing support set $\mathcal{S}'$. For each task, gradient descent is performed to adapt RL-Agent$_\theta$ (by Equations 4 and 5) and ML-Agent$_\eta$ (by Equations 9 and 10) with few-shot questions from $\mathcal{S}'$.

Testing Stage. As shown in Algorithm 4, each task T_i is performed using the corresponding adapted models (RL-Agent$_{\theta'_i}$ and ML-Agent$_{\eta'_i}$) on the testing query set $\mathcal{Q}'$. The average effectiveness, evaluated using a specific metric $\Delta(\cdot, \cdot)$, is reported across all tasks.

5 Experiments

5.1 Experimental Setup

Datasets. In line with previous research [19, 42], we conduct experiments on four widely-used task planning datasets: HotpotQA [38], ALFWorld [25], Webshop [39], and ScienceWorld [29]. HotpotQA is designed for multi-hop reasoning tasks, consists of approximately 113K QA pairs sourced from Wikipedia. ALFWorld enables agents to complete embodied tasks in a simulated environment (e.g., placing a washed apple in the kitchen fridge). Webshop is a web application that simulates an online shopping environment, where an agent navigates webpages to find, customize, and purchase an item based on a text instruction specifying the product requirements. ScienceWorld assesses agents' scientific reasoning abilities at the level of an elementary school science curriculum.

To set up the meta-learning, for HotpotQA, we define tasks using the 12 answer types in the dataset (e.g., Person, Location, Date), where we randomly select 6 types as the seen training tasks and 6 types as the unseen testing tasks. For ALFWorld, we use their provided seen tasks and unseen tasks for training and testing, respectively. For Webshop, we define tasks by product category, where we randomly sample 60% categories for training and the remaining for testing. For ScienceWorld, we utilize it to evaluate the generalization capability of `InstructRAG` across datasets, focusing on tasks that are entirely new to a trained `InstructRAG` model.

Baselines. We carefully review the literature and identify the following baseline methods: ReAct [41], WKM [19], Reflexion [23], GenGround [22], and RAP [11]. These correspond to recent representative techniques discussed in Section 2. For GenGround, we employ a retriever implemented by LlamaIndex [14] to find information that grounds LLM-generated answers, where we store TAO triplets from previous successful experiences across the datasets and utilize the retrieved similar triplets as the retriever's output. The same data (i.e., support sets from both training and testing tasks) is used to prepare the external database for the retrieval-based methods (i.e., GenGround and RAP). In addition, we incorporate the InstructRAG and baselines into three typical LLMs, namely GLM-4 [9], GPT-4o mini [1], and DeepSeek-V2 [7] for comparison.

To ensure fair performance comparisons, we note that: 1) Both the baselines and InstructRAG are configured with same setups, including the same retrievers and backbone LLMs; 2) we follow the hyperparameter settings specified in their original papers.

Evaluation Metrics. Following [15, 19, 42], we evaluate the effectiveness of InstructRAG on four datasets. For HotPotQA, the F1 score is used, comparing the agent's answers with the ground truth. For ALFWorld, the success rate, a binary metric (0 or 1), indicates whether the agent successfully completed the task. For WebShop and ScienceWorld, a reward score ranging from 0 to 1 is employed to measure the level of task completion. Overall, higher values indicate superior results. We note that all reported experimental results are statistically significant, verified by a t-test with $p < 0.05$.

Implementation Details. We implement InstructRAG and baselines using Python 3.7. The threshold δ for constructing instruction graphs is set to 0.4. In RL-Agent, we implement a two-layered feedforward neural network. The first layer consists of 20 neurons using the tanh activation function, and the second layer comprises 2 neurons corresponding to the action space to include or exclude a node. We use the Adam stochastic gradient descent with a learning rate of 0.001 to optimize the policy, and the reward discount is set to 0.99. In ML-Agent, the hyperparameter K for selecting a path is empirically set to 3. To boost training efficiency, we cache the inputs and outputs generated by the LLMs during training.

5.2 Experimental Results

(1) Effectiveness Evaluation (comparison with baseline methods). We evaluate the effectiveness of InstructRAG against baselines across three LLMs on unseen tasks in Table 1, InstructRAG consistently outperforms the baselines, demonstrating superior effectiveness. Notably, it achieves improvements of 19.2%, 9.3%, and 6.1% over the best baseline (RAP) on HotpotQA, ALFWorld, and Webshop, respectively. This improvement can be attributed to two factors: 1) InstructRAG employs a graph-based organization of instruction paths, enabling the combination into new paths for more effective planning, rather than independently storing them in an external database as RAP does, and 2) it utilizes a meta-learning approach to efficiently adapt the trained model to diverse tasks.
(2) Effectiveness Evaluation (generalization capabilities across datasets). We further evaluate the generalization capabilities of InstructRAG, by applying the trained InstructRAG model from HotpotQA to entirely new tasks in the ScienceWorld dataset. The results are reported in Table 2. Consistently, InstructRAG outperforms the best baseline method, RAP, with 6%-10% improvements.

Table 1: Effectiveness of InstructRAG on unseen tasks.

Backbone	Method	HotpotQA	ALFWorld	Webshop
GLM-4	ReAct [41]	24.04	47.01	62.13
	WKM [19]	-	64.18	68.14
	Reflexion [23]	26.88	52.99	67.91
	RAP [11]	27.86	64.18	69.45
	GenGround [22]	26.97	58.96	63.18
	InstructRAG	**33.61**	**72.39**	**71.25**
GPT-4o mini	ReAct [41]	25.45	42.54	49.16
	WKM [19]	-	55.22	54.56
	Reflexion [23]	27.39	50.75	51.31
	RAP [11]	27.66	56.71	56.31
	GenGround [22]	28.99	44.03	53.73
	InstructRAG	**31.05**	**58.21**	**64.18**
DeepSeek-V2	ReAct [41]	25.35	52.24	57.58
	WKM [19]	-	72.39	67.46
	Reflexion [23]	28.69	67.16	61.13
	RAP [11]	29.82	72.39	72.72
	GenGround [22]	33.50	67.16	62.24
	InstructRAG	**37.17**	**81.34**	**74.00**

Table 2: Effectiveness of InstructRAG across datasets (Training: HotpotQA, Testing: ScienceWorld).

HotpotQA → ScienceWorld	GLM-4	GPT-4o mini	DeepSeek-V2
RAP	24.37	23.49	32.15
InstructRAG	**26.85**	**25.10**	**33.96**

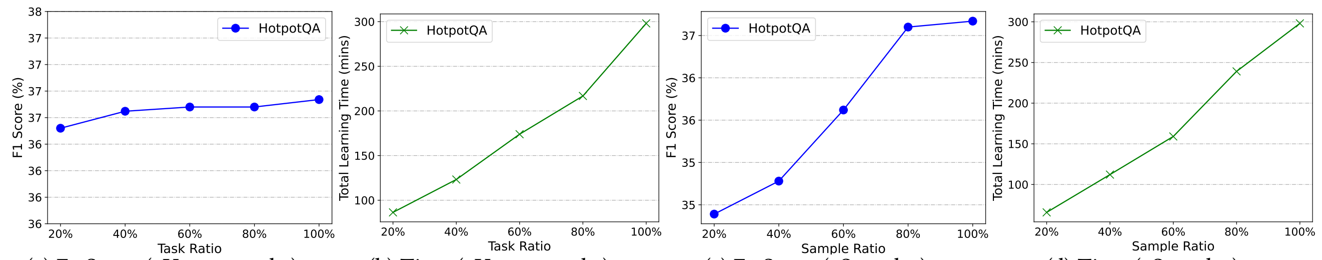
Table 3: Effectiveness of InstructRAG on seen tasks.

Backbone	Method	HotpotQA	ALFWorld	Webshop
GLM-4	RAP [11]	27.27	63.33	68.35
	InstructRAG	**34.99**	**72.50**	**73.60**
GPT-4o mini	RAP [11]	27.18	60.00	57.92
	InstructRAG	**31.09**	**64.17**	**64.92**
DeepSeek-V2	RAP [11]	31.53	75.83	71.01
	InstructRAG	**38.81**	**84.17**	**79.17**

Table 4: Robustness to erroneous historical paths.

Noise rate	0%	10%	20%	30%	40%	50%
RAP [11]	29.82	28.74	27.42	26.01	24.10	21.72
InstructRAG	37.17	36.64	36.17	35.45	34.29	33.04

(3) Effectiveness Evaluation (performance on seen training tasks). We also report the performance on seen training tasks. Compared to RAP, similar improvements are observed in Table 3, with average improvements of 21.9%, 10.8 %, and 10.4% on HotpotQA, ALFWorld, and Webshop, respectively.
(4) Effectiveness Evaluation (robustness to noise). We examine the impact of erroneous historical paths in the instruction graph on task performance, by introducing noisy paths (i.e., failed instruction paths from past experiences) with a controlled noise rate ranging from 0% to 50%. For comparison, we use RAP, the best baseline method, which also includes noisy paths in its database. The F1 score results, based on DeepSeek-V2 for HotPotQA, are reported in Table 4. Notably, even with a noise rate of 50%, the performance of InstructRAG remains relatively stable, with only a 11.1% decrease. This robustness stems from the diverse instruction combinations that help select appropriate paths and mitigate noise effectively.
(5) Ablation Study. We perform an ablation study to assess the contributions of different components within InstructRAG in Table 5.



(a) F1 Score (#Unseen tasks) (b) Time (#Unseen tasks) (c) F1 Score (#Samples) (d) Time (#Samples)

Figure 2: F1 scores and few-shot learning times wrt the number of unseen tasks or samples with DeepSeek-V2 on HotpotQA.

Table 5: Ablation study for verifying enlargeability (RL-Agent) and transferability (ML-Agent) on HotpotQA.

Components	F1 score
`InstructRAG`	**37.17**
w/o Instruction Graph	32.87
w/o RL-Agent	33.45
w/o warm-start in RL-Agent	34.37
w/o policy gradient in RL-Agent	36.18
w/o ML-Agent	34.78
w/o pre-training in ML-Agent	36.19
w/o fine-tuning in ML-Agent	36.24

Table 6: Impacts of threshold δ and runtime efficiency.

δ	0.0	0.2	0.4	0.6	0.8	1.0
F1 score	34.02	35.19	**37.17**	36.61	36.48	35.93
Construction (s)	19.61	21.27	**20.87**	21.65	23.08	22.07
Training (hours)	23.26	23.64	**23.93**	24.81	25.13	25.17
Few-shot (mins/task)	26.35	26.89	**27.10**	28.01	28.47	28.51
Testing (s)	33.87	34.46	**34.74**	35.85	36.45	36.47
# of nodes	5	29	**286**	666	720	725

Table 7: Impacts of the number of retrieved K paths.

K	1	2	3	4	5
F1 score	34.78	36.16	**37.17**	36.98	36.77
Testing (s)	32.05	33.57	**34.74**	35.31	42.09

We evaluate the following modifications: (1) omitting the instruction graph and allowing `InstructRAG` to retrieve relevant paths directly from stored individual paths; (2) omitting the RL-Agent and using a threshold-based method to determine node inclusion or exclusion, (3) the warm-start stage, (4) the policy gradient stage; (5) omitting the ML-Agent and relying solely on the path returned by the RL-Agent for testing on unseen tasks, (6) the pre-training stage, (7) the fine-tuning stage. We observe that the knowledge in the instruction graph contribute to a significant improvement of 11.6%, and both the RL-Agent and ML-Agent contribute to the overall improvements of 11.1% and 6.9%, respectively.

(6) Parameter Study (threshold δ for constructing instruction graphs and runtime efficiency). As shown in Table 6, we vary the threshold δ from 0.0 to 1.0 to control the graph construction process. As δ increases, more nodes are created, but the construction time remains stable. This is because the total number of indexed instructions with AKNN is not sensitive to the threshold. We observe that the F1 score initially increases and then decreases as δ increases. When $\delta = 0.0$, there are few instruction node sets to manage all instructions, making it difficult to accurately identify instructions for a given question due to the large set size. Conversely, when $\delta = 1.0$, the graph reduces to individual instruction paths, losing the flexibility to combine instructions into new paths. Therefore, a moderate δ leads to the best performance. In addition, we present the training, few-shot learning, and testing times as δ increases. Notably, training and few-shot learning require significantly more time than the graph construction, primarily due to the higher computational demands of language generation compared to the algorithmic construction. Furthermore, the graph construction is a one-time process conducted during data pre-processing.

(7) Parameter Study (the number of retrieved candidate paths K). We vary the number of retrieved paths K from 1 to 5 and report the F1 scores and testing times in Table 7. As expected, testing time increases with larger K due to the consideration of more candidate paths. We observe that overall performance converges when K reaches 3, at which point a potentially optimal path can be retrieved from the instruction graph.

(8) Impact of Few-shot Learning. `InstructRAG` includes a few-shot learning stage to quickly adapt to each task. We report its effectiveness and few-shot learning time based on the number of unseen tasks or the number of samples per task with DeepSeek-V2. As shown in Figure 2(a)-(b) , we vary the task ratio from 0.2 to 1.0, and observe that the effectiveness remains stable as the number of tasks increases, indicating a strong transferability across different tasks. The running time increases with more tasks due to the inclusion of additional training data. Additionally, we vary the sample ratio from 0.2 to 1.0 for each task. As shown in Figure 2(c) and Figure 2(d), we observe that the effectiveness improves and converges around 80% of the samples, while the running time increases as more samples are used for training. We note that, on average, a task requires 27.1 minutes for adaptation, and different tasks can be processed in parallel. The results for GLM-4 and GPT-4o mini show similar trends and are therefore omitted for brevity.

6 Conclusion

In this paper, we conduct a systematic study on leveraging RAG for task planning and identify two critical properties: enlargability and transferability. We introduce `InstructRAG`, a novel multi-agent meta-reinforcement learning solution that integrates an instruction graph, an RL-Agent, and an ML-Agent to optimize end-to-end task planning performance. Our extensive experiments on four widely used datasets, across various LLMs demonstrate that `InstructRAG` delivers superior performance and exhibits the ability to rapidly adapt to new tasks using few-shot examples. As a future direction, we plan to extend `InstructRAG` to accommodate more tasks.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI*, Vol. 38. 17682–17690.

[3] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915* (2023).

[4] Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. 2022. Meta-learning via Language Model In-context Tuning. In *ACL*. 719–730.

[5] Gautier Dagan, Frank Keller, and Alex Lascarides. 2023. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391* (2023).

[6] Budhaditya Deb, Ahmed Hassan, and Guoqing Zheng. 2022. Boosting Natural Language Generation from Instructions with Meta-Learning. In *EMNLP*. 6792–6808.

[7] DeepSeek-AI. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. *arXiv preprint arXiv:2405.04434* (2024).

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*. PMLR, 1126–1135.

[9] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. *arXiv preprint arXiv:2406.12793* (2024).

[10] Malte Helmert. 2006. The fast downward planning system. *JAIR* 26 (2006), 191–246.

[11] Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. 2024. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *arXiv preprint arXiv:2402.03610* (2024).

[12] Myeonghwa Lee, Seonho An, and Min-Soo Kim. 2024. PlanRAG: A Plan-then-Retrieval Augmented Generation for Generative Large Language Models as Decision Makers. *arXiv preprint arXiv:2406.12430* (2024).

[13] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477* (2023).

[14] Jerry Liu. 2022. *LlamaIndex*. doi:10.5281/zenodo.1234

[15] Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. 2023. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960* (2023).

[16] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *NeurIPS* 36 (2024).

[17] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *TPAMI* 42, 4 (2018), 824–836.

[18] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943* (2021).

[19] Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024. Agent Planning with World Knowledge Model. *arXiv preprint arXiv:2405.14205* (2024).

[20] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *ICML*. PMLR, 8748–8763.

[21] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. 2022. Planning with large language models via corrective re-prompting. In *NeurIPS Workshop*.

[22] Zhengliang Shi, Shuo Zhang, Weiwei Sun, Shen Gao, Pengjie Ren, Zhumin Chen, and Zhaochun Ren. 2024. Generate-then-Ground in Retrieval-Augmented Generation for Multi-hop Question Answering. *ACL* (2024).

[23] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS* 36 (2024).

[24] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10740–10749.

[25] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768* (2020).

[26] Sanchit Sinha, Yuguang Yue, Victor Soto, Mayank Kulkarni, Jianhua Lu, and Aidong Zhang. 2024. MAML-en-LLM: Model agnostic meta-training of LLMs for improved in-context learning. *KDD* (2024).

[27] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502* (2024).

[28] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[29] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. ScienceWorld: Is your Agent Smarter than a 5th Grader?. In *EMNLP*. 11279–11298.

[30] Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. 2024. Learning From Failure: Integrating Negative Examples when Fine-tuning Large Language Models as Agents. *arXiv preprint arXiv:2402.11651* (2024).

[31] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[32] Yizhong Wang, Swaroop Mishra, et al. 2022. Benchmarking generalization via in-context instructions on 1,600+ language tasks. *arXiv preprint arXiv:2204.07705* 2 (2022).

[33] Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. 2024. Rat: Retrieval augmented thoughts elicit context-aware reasoning in long-horizon generation. *arXiv preprint arXiv:2403.05313* (2024).

[34] Zhaowei Wang, Hongming Zhang, Tianqing Fang, Ye Tian, Yue Yang, Kaixin Ma, Xiaoman Pan, Yangqiu Song, and Dong Yu. 2024. DivScene: Benchmarking LVLMs for Object Navigation with Diverse Scenes and Objects. *arXiv preprint arXiv:2410.02730* (2024).

[35] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS* 35 (2022), 24824–24837.

[36] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[37] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* (2023).

[38] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *EMNLP*. 2369–2380.

[39] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *NeurIPS* 35 (2022), 20744–20757.

[40] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS* 36 (2024).

[41] 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).

[42] Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Ningyu Zhang, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. 2024. Knowagent: Knowledge-augmented planning for llm-based agents. *arXiv preprint arXiv:2403.03101* (2024).