AdaptJobRec: Enhancing Conversational Career Recommendation through an LLM-Powered Agentic System

Qixin Wang[1,2], Dawei Wang[1], Kun Chen[1], Yaowei Hu[1]
Puneet Girdhar[1], Ruoteng Wang[1], Aadesh Gupta[1], Chaitanya Devella[1]
Wenlai Guo[1], Shangwen Huang[1], Bachir Aoun[1], Greg Hayworth[1]
Han Li[1]*, Xintao Wu[2]*
[1]Walmart Global Tech
[2]University of Arkansas
han.li@walmart.com, xintaowu@uark.edu

Abstract

In recent years, recommendation systems have evolved from providing a single list of recommendations to offering a comprehensive suite of topic-focused services. To better accomplish this task, conversational recommendation systems (CRS) have progressed from basic retrieval-augmented LLM generation to agentic systems with advanced reasoning and self-correction capabilities. However, agentic systems come with notable response latency—a longstanding challenge for conversational recommendation systems. To balance the trade-off between handling complex queries and minimizing latency, we propose AdaptJobRec, the first conversational job recommendation system that leverages autonomous agent to integrate personalized recommendation algorithm tools. The system employs a user query complexity identification mechanism to minimize response latency. For straightforward queries, the agent directly selects the appropriate tool for rapid responses. For complex queries, the agent uses the memory processing module to filter chat history for relevant content, then passes the results to the intelligent task decomposition planner, and finally executes the tasks using personalized recommendation tools. Evaluation on Walmart's real-world career recommendation scenarios demonstrates that AdaptJobRec reduces average response latency by up to 53.3% compared to competitive baselines, while significantly improving recommendation accuracy.

Introduction

In recent years, the complexity of conversational recommendation systems (CRS) has been continuously increasing (Zhang et al. 2024; Huang et al. 2023; Fang et al. 2024), evolving from offering a single type of recommendation to providing a range of services on specific topics. To improve task completion rates, these systems have progressed from LLM-based retrieval-augmented generation (RAG) models (Friedman et al. 2023; Gao et al. 2023a; Liu et al. 2023; Kuzi and Malmasi 2024) to sophisticated agentic systems with advanced reasoning and self-correction capabilities. Such systems dynamically integrate multiple modules (e.g., planning and memory processing) and various tools (e.g., databases, search engines, and knowledge graphs) to better serve users' needs (Zhang et al. 2024). For instance, the ReAct architecture (Yao et al. 2023), developed by Princeton University and the Google Brain team, is a widely used agentic architecture in the industry. It employs a reasoning agent that leverages Chain-of-Thought (CoT) to decompose a user query into multiple sub-tasks, invoking corresponding tools to obtain information and complete tasks based on the feedback step by step. Another widely used architecture, the Plan and Execute agent (Wang et al. 2023a) incorporates a separate replan module to assess task completion quality and generate the subsequent steps, enhancing the model's reliability in handling complex problems. Other innovative approaches include the RecMind agentic recommendation system (Wang et al. 2023b) designed by the Amazon Alexa AI team, which proposes a novel Self-Inspiring Planning module that considers previously explored states in Tree of Thought (ToT) while planning the next step, effectively improving the planning ability of the agentic system, and the MACRS Agentic system (Fang et al. 2024), which employs an Asking Responder Agent to elicit user preferences through additional dialogue rounds before generating recommendations. Meanwhile, InteRecAgent (Huang et al. 2023) proposed by Microsoft Research Asia improves long-conversation handling with components such as a memory bus, dynamic demonstration-augmented task planning, and reflective processing.

Although these planning, memory, and replan modules significantly enhance an agentic system's reasoning capability and problem-solving ability, they also increase response latency. To address this challenge, researchers in JD eCommence proposed the concept of first token latency (Nie et al. 2024), demonstrating that an agentic CRS supported by a fine-tuned LLM can reduce the time required to output the initial response token, making an important contribution in this area.

Solving the response latency issue is particularly critical in contexts like Walmart's conversational job recommendation system. Walmart is a leading Fortune 500 company with millions of job applicants per year, where many users require rapid access to simple information, such as application status or interview time. In such cases, users are concerned with

*corresponding author

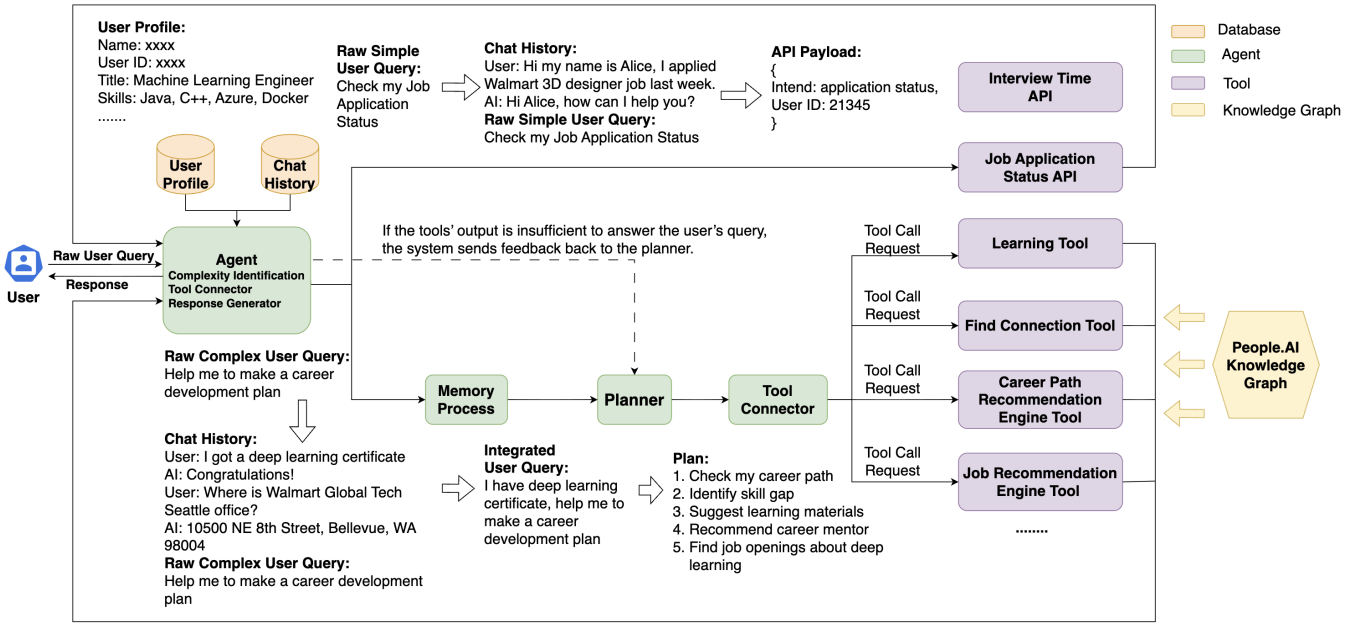

Figure 1: Architecture of AdaptJobRec Agentic System

when they receive the needed information, merely reducing first token latency is insufficient. To overcome the delays of agentic systems, we propose the AdaptJobRec architecture. This architecture integrates a complexity identification mechanism that distinguishes between simple and complex user queries. It employs the memory processing and task decomposition planner exclusively for complex queries, ensuring rapid responses for simple queries while maintaining the capability to handle complex user needs. We compare AdaptJobRec against two fine-tuned LLMs career path models (Liu et al. 2024; Touvron et al. 2023) and three widely used agentic systems (Wang et al. 2023a; Yao et al. 2023; Fang et al. 2024), demonstrating that AdaptJobRec outperforms these methods in prediction accuracy while delivering lower response latency.

Furthermore, besides reducing response latency in conversational recommendation systems, an equally important goal is minimizing the number of dialogue rounds needed for users to obtain key information. To achieve this goal, we develop a novel few-shot memory processing module that filters chat history for content relevant to the current user query, eliminating redundant planning, boosting planner accuracy, and enabling more precise tool selection. We also developed an intelligent task decomposition planner capable of generating a nested sub-task list, grouping tasks that can be executed asynchronously into the same sub-list. The AdaptJobRec agentic conversational recommendation system integrates memory processing, a task decomposition planner, and personalized recommendation tools based on user profiles and behavior. These components reduce the ask-back rounds needed to clarify user needs and the number of follow-up queries after receiving ineffective recommendations.

Methodology

Architecture of AdaptJobRec

As depicted in Figure 1, AdaptJobRec comprises several key components: an agent with a complexity identification mechanism, a few-shot learning memory process module, a task decomposition planner, and recommendation tools powered by the Walmart People.AI knowledge graph. This knowledge graph includes 1.6 million nodes and 83 million edges, representing entities such as job titles, openings, associates, applicants, and skills etc. This knowledge graph serves as the foundation for a series of personalized recommendation tools. The AdaptJobRec agent quickly addresses simple queries using different tools, while complex queries trigger the memory process module and task decomposition planner to activate a suite of personalized recommendation tools to generate a high quality response.

In conversational recommendation systems, a user's intent is rarely fully captured from their initial query. To deliver more accurate recommendations, it is essential to consider not only the query itself but also the user profile and behavior. For instance, consider a senior software engineer with six years of Java programming experience who states, "Find me some manager positions in the Seattle area; if none are available, other big cities will do." In this case, suggesting a Walmart store manager role in Seattle would be inappropriate, while recommending a lead software engineer position in New York that requires over five years of Java experience would better meet the user's needs, even if the former is closer to the original user query in the semantic vector space. Moreover, taking user behavior into account can further enhance recommendation quality. For example, if a senior software engineer recently saved three eCommerce related job openings on the Walmart careers website,

it strongly indicates an interest in that field. Even without an explicit mention in the conversation, the recommendation system should proportionally increase the presence of eCommerce job openings in its results. Based on this idea, several tools in AdaptJobRec have been designed as personalized recommendation engines powered by the People.AI knowledge graph that replaces traditional search engines.

Complexity identification mechanism

The Walmart job conversational recommendation system categorizes user queries into two types. The first type includes simple and clear queries, such as "help me check job application status", which require a fast and straightforward response. The second type involves more complex or ambiguous queries, such as "can you create a career development plan for me?", which should be decomposed into multiple sub-tasks: 1) Recommend a career path based on the user's current position, skills, and career interests; 2) Identify the skill gap between the user and the potential next role; 3) Suggest learning resources based on the identified skill gap; 4) Recommend a career mentor based on the skill gap and potential next role; 5) Suggest job openings that the user can apply for. The system then invokes the necessary tools to complete each sub-task and synthesizes the results to generate a comprehensive response.

If we were to handle all types of user queries using an agent that includes modules such as a planner and a memory process module, the agentic CRS would respond too slowly to the simpler user queries, negatively impacting user experience. Therefore, we have designed an agentic system with a user query complexity identification mechanism, which is implemented using a customized system prompt. As shown in Figure 2, the key parts of this system prompt that enable it to distinguish between simple and complex queries are highlighted in red. When a simple user query is received, the agent merges the user query with chat history without additional process and directly leverages tools such as the personalized recommendation APIs, or the People.AI knowledge graph to respond rapidly. For a complex query, the agent forwards the user query and additional relevant information (such as chat history, user profile, and career interests) to a few-shot learning memory processing module that integrates only the relevant memories to create an enhanced query, which is then decomposed into sub-tasks by a planner. The agentic system then employs the appropriate tools corresponding to tasks to fetch information, such as invoking the job recommendation engine, career path recommendation engine, or converting a sub-task into a Cypher query for the People.AI knowledge graph, and ultimately synthesizes them to provide a comprehensive response. When the information provided by the tool is not sufficient to satisfy the user's query, the system sends feedback to the planner, initiates a new planning cycle, and repeats the tool selection process. This process is indicated by the dashed arrow in Figure 1. This optimization can significantly reduce the latency of the LLM agentic system for simple queries while ensuring the accuracy of responses to complex and challenging user queries.

```
Assistant is a large language model designed to be
helpful, honest, and harmless. For straightforward user
queries, leverage available tools to provide concise
and accurate responses. For complex queries, engage the
memory processing module and task planner to
systematically break down and address all aspects of
the request while ensuring clarity and precision.

Assistant has access to the following tools:
Job_recommendation: A personalize recommendation engine
that can provide job openings to user.
Interview_time: response the interview time.
……
Career_path: A career path recommendation engine.
Complex_task: to finish complex task, include memory
processing module and task planner.

Use the following format:

Question: the input question you must answer
Thought: I should think step by step to determine what
to do
Action: the action to take, should be one of [search]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can
repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input
question

Begin!

Question: {input}
{agent_scratchpad}
```

Figure 2: System Prompt of Complexity Identification Mechanism.

Memory processing module

While techniques like windowing (Dai et al. 2019), summarization (Wang et al. 2025), and VectorDB (Hatalis et al. 2023) effectively compress chat history and provide the necessary context for a conversation, they often omit important details or retain irrelevant content. Such imprecision undermines the planner's ability to decompose complex queries in an agentic system. To improve the handling of complex queries in the AdaptJobRec agentic system, we design a series of few-shot learning examples covering various scenarios to build a memory processing module that enables LLM to precisely extract chat history segments relevant to the current user query.

Figure 3 presents parts of the system prompt used by the memory process module. In few-shot example 1, chat history segments relevant to the current user query (highlighted in green) are extracted and merged with the current user query (highlighted in purple) under the [TEXT] section (irrelevant segments appear in blue). The resulting Integrated User Query is shown in orange after [OUTPUT]. Few-shot example 2 illustrates a scenario where the chat history contains no content pertinent to the current query.

Planner

We leverage the planner, which is a commonly used component in LLM-powered agentic systems, to decompose Integrated User Query (from the memory processing module) into sub-tasks and execute them with appropriate tools.

```
You need to re-write the user request on the
last part of the text under [TEXT]
while considering the chat history if additional
information is needed.
For example:

[TEXT]
User:Where is the Bentonville Walmart Fitness
Center located?
Assistant: 1400 SE 5th St, Bentonville, AR 72716
User: How many people applied for the 3D
Designer job last week?
Assistant: 512 candidates applied for the 3D
Designer job last week.
User: how about the Graphic Designer role?
[OUTPUT]
How many people applied for the Graphic Designer
job in last week?

[TEXT]
User: How many people apply 3D Designer job in
last week?
Assistant:512 candidates apply 3D designer job
in last week.
User: How's the weather today?
[OUTPUT]
How's the weather today?

[TEXT]
{chat_history}
{user_query}
[OUTPUT]
```

Few-shot example 1: Chat history contains relevant information

Few-shot example 2: No relevant information in chat history

Figure 3: System Prompt of Few-shot Learning Memory Processing Module.

In existing LLM-powered agentic systems (Wang et al. 2023a,b) such as the Plan & Execute Agent and RecMind Agent, planners typically generate a strictly ordered sequence of sub-tasks, invoking tools sequentially. However, while certain tasks inherently require sequential execution, many can be executed asynchronously, offering the potential to reduce response latency.

For example, given the query "Which city has more machine learning engineer job openings, Seattle or Sunnyvale?", the planner may generate the following task sequence:

- ['Get machine learning engineer job opening number from Seattle', 'Get machine learning engineer job opening number from Sunnyvale']
- ['Compare job opening numbers of Sunnyvale and Seattle']

The first two retrieval operations can be executed in parallel, followed by the comparison step.

Therefore, in AdaptJobRec, we enhance the existing planner to output a nested list, grouping asynchronously executable sub-tasks within the same sub-list. This functionality is implemented via a few-shot learning module, with part of the system prompt shown in Figure 4. The prompt explicitly instructs the LLM to group sub-tasks that can be executed concurrently (highlighted in red). Few-shot example 1 shows an Integrated User Query containing two sub-tasks that can be executed in parallel (highlighted in purple). Few-shot example 2 illustrates a case with no asynchronously executable sub-tasks.

```
You need to split the sentences under [TEXT]
into several sub-tasks, following the format
shown in the examples. Group the subtasks that
can be executed asynchronously into a single
list.
If the sentence contains only one intent, do not
split it.

For example,

[TEXT]
Which city has more machine learning engineer
job openings, Seattle or Sunnyvale?
[OUTPUT]
- [Get machine learning engineer job opening
number from Seattle; Get machine learning
engineer job opening number from Sunnyvale]
- [Compare job opening numbers of Sunnyvale and
Seattle]

[TEXT]
How many machine learning engineer job openings
at Sunnyvale?
[OUTPUT]
- [Get machine learning engineer job opening
number from Sunnyvale]

[TEXT]
{integrated_user_query}
[OUTPUT]
```

Few-shot example 1: Contain asynchronously executable sub-tasks

Few-shot example 2: Do not contain asynchronously executable sub-tasks

Figure 4: System Prompt of AdaptJobRec Planner.

Tools

Personalized Job Recommendation Tool When a user starts a conversation, the personalized job recommendation engine extracts their current job title, queries the People.AI knowledge graph for adjacent positions, and selects active openings with the most recent posting dates. It then matches key entities (e.g., skills, education, location) from the user profile to each candidate job opening, computes and sums similarity scores, and normalizes the sum score by the total number of entities. A user interest function then adjusts the normalized score based on real-time interactions (e.g., clicks, saves, likes, dislikes) with job openings across different job families. Both entity match weights and interest function parameters are tuned by Bayesian optimization on click data to maximize Click Through Rate (CTR), and the top 20 ranked results are forwarded to the main agent to generate the response.

Career Path Recommendation Tool Career path recommendations fall into two categories. In one, the user explicitly states a desired destination, for example, "I want to become a principal 3D designer; what should I do?" In the other, the destination is not specified, for instance, "I just received a job offer for the Walmart Merchant position; please show me the future development prospects for this role." For the first scenario, our career path recommendation tool ex-

tracts the user's current position from their profile and applies a shortest path algorithm using edge weights from the people.AI knowledge graph to identify and suggest potential career paths. For the second scenario, we develop a personalized career path growth algorithm that recommends several tailored career paths. This algorithm considers the user's current position, skill set, and domain preferences, gleaned from recent activity, to comprehensively illustrate the role's future development.

Cypher Tools We develop two types of Cypher Tools. The first consists of predefined Cypher code templates, where the agent selects the appropriate tool based on the user's intent and fills in key entities, minimizing response time for high frequency queries. The second is a more flexible Text-to-Cypher approach, which feeds both the knowledge graph schema and user input into the LLM to generate Cypher queries, allowing it to address a broader range of user queries.

Application Deployment

As shown in Figure 5, AdaptJobRec is implemented as a set of independently deployable services, each responsible for a distinct function in query handling and recommendation generation. The Front End provides the user interface and communicates with the backend via Kafka, which serves as the streaming backbone for publishing user queries and delivering agent responses. The AdaptJobRec Server orchestrates backend operations, retrieving user information from the User Profile Service API and historical conversations from the Cassandra database, while leveraging Redis to cache frequently accessed query results and tool responses to reduce latency. The AdaptJobRec Agent hosts the LLM-based reasoning and decision logic, which classifies queries into simple or complex. Simple queries are routed to the Job Application Microservice, while complex queries are handled through the Model Context Protocol (MCP) (Hou et al. 2025) server, which call recommendation engine tools or execute Cypher queries against the People.AI Knowledge Graph stored in the graph database.

Upon receiving a user query, the Front End publishes the corresponding topic to the Kafka cluster (1 in Figure 5). The AdaptJobRec Server, subscribed to this topic, retrieves the query (2 in Figure 5) and uses the login credentials provided by the Front End to call the User Profile Service API (2a in Figure 5), obtaining personalized user information. In parallel, it fetches the user's conversation history from the Cassandra database (2b in Figure 5).

To reduce response latency for high frequency queries (e.g., "What can be the future role of a Walmart cashier?"), we integrate Redis with the AdaptJobRec Server to enable Cache Augmented Generation (3a in Figure 5). This mechanism stores and reuses tool call responses for frequently asked queries, thereby eliminating redundant LLM calls and reducing overall response time.

Within the AdaptJobRec Agent, the complexity identification mechanism evaluates the user query, profile, and chat history (3 in Figure 5) to classify the request as either simple or complex. For simple queries (4a in Figure 5), the agent directly accesses the Job Application Microservice, which provides a suite of APIs to retrieve the information needed for response generation. For complex queries (4), the query and profile are routed through the memory process module and planner component before the agent extracts key entities and formulates a tool invocation request to the MCP Server. The MCP Server then either executes recommendation tools (5a in Figure 5) or generates a Cypher query (5 in Figure 5) to retrieve the required information from the People.AI Knowledge Graph (6 in Figure 5), and returns the results to the agent (7 in Figure 5).

Finally, the AdaptJobRec Agent synthesizes the final answer based on information retrieved from either the Job Application Microservice or the MCP Server and streams the response to Kafka (8 in Figure 5). The Front End, acting as a Kafka consumer (9 in Figure 5), processes this stream and delivers the final response to the user.

Methods	Hit@10	NDCG@10	MAP@10
RAG	0.200	0.039	0.014
ReAct	0.279	0.075	0.034
Plan & Execute	0.313	0.080	0.036
MACRS	0.312	0.078	0.035
AdaptJobRec	**0.318**	**0.081**	**0.037**

Table 1: Comparison on Job Recommendation Task

AdaptJobRec vs. Baseline	p-value (Hit@10)	p-value (NDCG@10)	p-value (MAP@10)
RAG	$<$0.001	$<$0.001	$<$0.001
ReAct	$<$0.001	$<$0.001	$<$0.001
Plan & Execute	0.001	0.008	0.027
MACRS	0.001	0.008	0.028

Table 2: Testing Result Comparing **AdaptJobRec** with Baselines on Job Recommendation Task

Evaluation Result

In this section, we evaluate the performance of AdaptJobRec on three tasks: Job Recommendation, Career Path Prediction, and a Pilot User Study using real-world data from Walmart company.

Job Recommendation

We collected user profiles, browsing histories, and click data from 10,014 users interacting with the Walmart Job Recommendation System in 2024, and subsequently compared the performance of AdaptJobRec, LLM RAG (Gao et al. 2023b), the ReAct Agent (Yao et al. 2023), MACRS Agent (Fang et al. 2024), and the Plan & Execute Agent (Wang et al. 2023a) on job recommendation tasks.

Table 1 shows that AdaptJobRec consistently outperforms all four baselines, across metrics Hit@10, NDCG@10, and MAP@10, achieving the highest scores of 0.3176, 0.0810, and 0.0371, respectively. Compared with the strongest baseline (Plan & Execute), AdaptJobRec still delivers measurable improvements in all metrics. As confirmed by Welch's

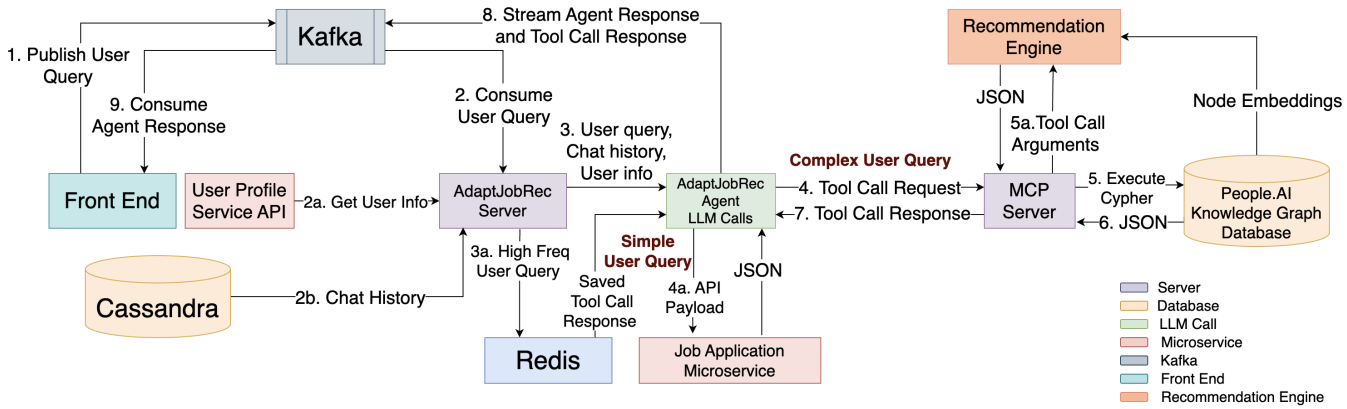

Figure 5: AdaptJobRec Agentic System Deployment Architecture

Methods	% Hit Real Trans ↑	Latency(s) ↓
Frequency	8.12	0.32
Llama-Capa	8.35	0.67
DeepSeek-Capa	11.24	0.81
AdaptJobRec	**12.82**	**0.36**

Table 3: Comparison on Career Path Prediction Task

AdaptJobRec vs. Baseline	p-value (Hit Real Trans)	p-value (Latency)
Frequency	0.017	–
Llama-Capa	0.018	0.021
DeepSeek-Capa	0.031	0.025

Table 4: Testing Result Comparing **AdaptJobRec** with Baselines on Career Path Prediction Task

t-test in Table 2, these gains are statistically significant, highlighting AdaptJobRec's robust advantage in both retrieval accuracy and ranking quality for job recommendation tasks.

Career Path Prediction

We collected 932,854 Walmart associate job transition records from 2022 to 2023 as training data for AdaptJobRec. We fine-tuned Llama-3.1-8B (Touvron et al. 2023) and DeepSeek-R1-Distill-Qwen-7B (Liu et al. 2024) on this dataset, yielding the Llama-Capa and DeepSeek-Capa models. We then used 471,495 job transition records from Walmart associates in 2024 as testing data to compare AdaptJobRec's performance against the two fine-tuned LLM models, Llama-Capa and DeepSeek-Capa.

As shown in Table 3 and Table 4, AdaptJobRec attains the highest hit rate of real transitions (12.82%) among all methods, significantly surpassing Frequency, Llama-Capa, and DeepSeek-Capa. Despite its superior accuracy, AdaptJobRec maintains a low response latency (0.36 s), significantly faster than Llama-Capa and DeepSeek-Capa and

Methods	Conv Round ↓	Resp Latency (ms) ↓
RAG	7.10	1065
Plan & Execute	6.38	957
ReAct	3.86	579
MACRS	3.68	552
AdaptJobRec	**3.32**	**498**

Table 5: Comparison of Average Conversation Rounds and Response Latency for the Pilot Group

AdaptJobRec vs. Baseline	p-value (Conv Round)	p-value (Latency)
RAG	<0.001	<0.001
ReAct	<0.001	<0.001
Plan & Execute	<0.001	<0.001
MACRS	0.002	<0.001

Table 6: A/B Testing Result Comparing **AdaptJobRec** with Baselines on Average Conversation Rounds and Response Latency for the Pilot Group

close to the minimal latency Frequency method. These results demonstrate that AdaptJobRec achieves an effective balance between accuracy and efficiency for career path prediction.

Pilot Group Study

We logged 150 conversation sessions from a pilot group of 30 users. At the start of each session, the experimental UI randomly selected one of four methods: the ReAct Agent (Yao et al. 2023), the Plan & Execute Agent (Wang et al. 2023a), MACRS Agent (Fang et al. 2024), or AdaptJobRec. We measured the average number of conversation rounds needed to obtain the target information and the total session response time for each user. Conversation rounds of sessions in which users hadn't acquired the information by the end of the conversation were assigned as 20.

Table 5 and Table 6 present the comparison and Welch's

t-test results between AdaptJobRec and four baseline methods (RAG, Plan & Execute, ReAct, and MACRS) on two key metrics: (1) the average number of conversation rounds per session (2) the average response latency. As shown in Table 5, AdaptJobRec achieves the lowest average conversation rounds (3.32) and the fastest response latency (498 ms), representing a 54% reduction in conversation rounds and a 53% improvement in latency compared to the RAG baseline. Welch's t-test results in Table 6 confirm that AdaptJobRec's improvements over the baselines are statistically significant.

During the pilot study, users intentionally asked large percentage of complex queries to test the system, leaving simple queries at under 5% of requests. This limited AdaptJobRec's ability to showcase its lower response latency. After the application launch, real world usage will include many more simple queries, making its latency advantage much more apparent.

Conclusion and Future Work

In this work, we present AdaptJobRec, an LLM-powered agentic conversational job recommendation system that handles diverse career recommendation tasks. AdaptJobRec incorporates a complexity identification mechanism, a memory processing module, and a task decomposition planner with multiple personalized recommendation tools. Simple queries are routed directly to the appropriate tool for rapid responses. Complex queries are first handled by the memory processing module and then passed to the Intelligent task planner. These processes eliminates redundant planning and boosts the planner's tool selection accuracy. Evaluation across varied scenarios shows that AdaptJobRec significantly reduces latency without sacrificing accuracy. Future work will broaden its personalized toolset and further refine each module's performance.

References

Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q. V.; and Salakhutdinov, R. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Fang, J.; Gao, S.; Ren, P.; Chen, X.; Verberne, S.; and Ren, Z. 2024. A multi-agent conversational recommender system. *arXiv preprint arXiv:2402.01135*.

Friedman, L.; Ahuja, S.; Allen, D.; Tan, Z.; Sidahmed, H.; Long, C.; Xie, J.; Schubiner, G.; Patel, A.; Lara, H.; et al. 2023. Leveraging large language models in conversational recommender systems. *arXiv preprint arXiv:2305.07961*.

Gao, Y.; Sheng, T.; Xiang, Y.; Xiong, Y.; Wang, H.; and Zhang, J. 2023a. Chat-rec: Towards interactive and explainable llms-augmented recommender system. *arXiv preprint arXiv:2303.14524*.

Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, H.; and Wang, H. 2023b. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2.

Hatalis, K.; Christou, D.; Myers, J.; Jones, S.; Lambert, K.; Amos-Binks, A.; Dannenhauer, Z.; and Dannenhauer, D. 2023. Memory matters: The need to improve long-term memory in llm-agents. In *Proceedings of the AAAI Symposium Series*, volume 2, 277–280.

Hou, X.; Zhao, Y.; Wang, S.; and Wang, H. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*.

Huang, X.; Lian, J.; Lei, Y.; Yao, J.; Lian, D.; and Xie, X. 2023. Recommender ai agent: Integrating large language models for interactive recommendations. *arXiv preprint arXiv:2308.16505*.

Kuzi, S.; and Malmasi, S. 2024. Bridging the gap between information seeking and product search systems: Q&A recommendation for e-commerce. In *ACM SIGIR Forum*, volume 58, 1–10. ACM New York, NY, USA.

Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liu, Y.; Zhang, W.-N.; Chen, Y.; Zhang, Y.; Bai, H.; Feng, F.; Cui, H.; Li, Y.; and Che, W. 2023. Conversational recommender system and large language model are made for each other in E-commerce pre-sales dialogue. *arXiv preprint arXiv:2310.14626*.

Nie, G.; Zhi, R.; Yan, X.; Du, Y.; Zhang, X.; Chen, J.; Zhou, M.; Chen, H.; Li, T.; Cheng, Z.; et al. 2024. A hybrid multi-agent conversational recommender system with llm and search engine in e-commerce. In *Proceedings of the 18th ACM Conference on Recommender Systems*, 745–747.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Wang, L.; Xu, W.; Lan, Y.; Hu, Z.; Lan, Y.; Lee, R. K.-W.; and Lim, E.-P. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Wang, Q.; Fu, Y.; Cao, Y.; Wang, S.; Tian, Z.; and Ding, L. 2025. Recursively summarizing enables long-term dialogue memory in large language models. *Neurocomputing*, 130193.

Wang, Y.; Jiang, Z.; Chen, Z.; Yang, F.; Zhou, Y.; Cho, E.; Fan, X.; Huang, X.; Lu, Y.; and Yang, Y. 2023b. Recmind: Large language model powered agent for recommendation. *arXiv preprint arXiv:2308.14296*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhang, X.; Xie, R.; Lyu, Y.; Xin, X.; Ren, P.; Liang, M.; Zhang, B.; Kang, Z.; de Rijke, M.; and Ren, Z. 2024. Towards empathetic conversational recommender systems. In *Proceedings of the 18th ACM Conference on Recommender Systems*, 84–93.