TripChat: A Long-term Planning Agent Hierarchy for Travel

Lutfi Eren Erdogan, Alp Eren Ozdarendeli, Nathaniel Haynam, Chuyi Shang, Inga Zhuravleva
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA, USA
{lerdogan, alp_ozdarendeli, natehaynam, chuyishang, ing4}@berkeley.edu

***Abstract*— In this work, we introduce a zero-shot, unsupervised trip and travel planner. Planning a trip automatically has several use cases - it can save people time and money, help people create better itineraries, and provide inspiration. Our framework uses a hierarchical multi-agent approach to coordinate, gather real-world data using APIs, and plan out a travel plan. Existing AI-based approaches for travel planning are often too simplistic and inflexible, rely on outdated data, or are of poor quality and detail. Our results show that our output plans generated this way are comparable to the best human made plans, approaching similar levels of detail, experiences, and personalization.**

Keywords—Artificial intelligence (AI), Function calling, Large language models, Machine Learning (ML), Multi-agent hierarchy

I. Introduction

A fundamental difficulty for travellers in the modern era of travel is the lack of a centralized, real-time system that provides individualized suggestions. This gap in the travel planning process is multifaceted, encompassing aspects such as local events, flight bookings, car rentals, cultural insights, and weather updates. The complexity of planning trips, especially those involving multiple destinations or activities, further worsens the issue. A solution is needed not only to improve the trip experience but also to streamline the decision-making process, therefore lowering the time and effort engaged in travel preparation. This need becomes more pronounced when considering the dynamic nature of travellers' preferences and real-world availability, requiring the system to be robust against these changes. To address this challenge, we introduce TripChat, a novel solution designed to generate efficient and tailored travel plans. TripChat leverages a hierarchical agent structure implemented with Microsoft's Autogen [4] framework for multi-agent systems. This structure allows for specialized agents, each with distinct responsibilities, to collaboratively create comprehensive travel plans. The system's architecture is built on the principles of intelligent route modification, comprehensive planning, and real-time adaptability. It can complete a wide range of travel requests, from simple day trips to complex multi-stop excursions. An entire trip plan is generated in under 4 minutes. Our code can be found here: https: //github.com/xTRam1/TripChat

II. Methodology

A. Agent Hierarchy

We designed a hierarchical agent structure where agents of different responsibilities work together for the common goal of creating a trip plan. In other words, the tasks of finding local events, hotels, flights, etc. are abstracted out into different teams and agents. For this, we use Microsoft's Autogen [4] framework for multi-agent systems. We use its GroupChat framework, which enables the agents in a system to talk with each other for collaboration. A GroupChat works in "rounds."

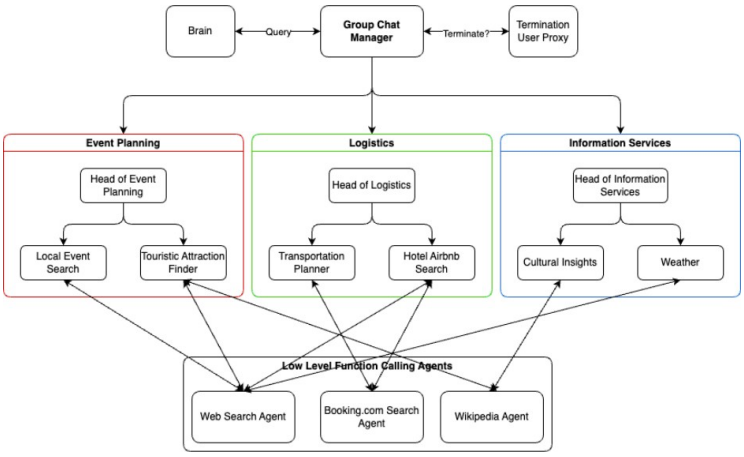

Figure 1: A Diagram of the Agent Hierarchy

In each round, the group chat manager selects a speaker to talk and contribute to the conversation, according to the past chat memory, the user query, and its reasoning capabilities. In our implementation, we gave the agents themselves the ability to "suggest" the next speaker to talk to create a hierarchy structure.

This hierarchy structure can be found in Fig. 1.

All the agent system prompts are designed so that everyone is aware of its team members' capabilities and who they are allowed to talk to. We implemented this kind of structure by using the "function-calling" capabilities of

Manuscript Received: 02.03.2024 | Manuscript Accepted: 22.03.2024 | DOI: 10.5281/zenodo.10938300

LLMs: In reality, only low-level agents actually call the tools that interact with the outside world (such as interacting with the web, with travel APIs like booking.com search, and Wikipedia); whereas, the "functions" of the team members are functions that used for calling these low-level agents. The low-level agents are implemented with the Langchain framework. For a more detailed description of how a group chat proceeds, please see the Group Chat Implementation section.

B. Why Hierarchy

Continuing from the previous description of our hierarchical agent structure, we now delve into the rationale behind choosing a multi-agent setup instead of a single agent with access to all APIs. This decision is driven primarily by considerations of scalability and context length management. In a single-agent framework with access to a wide array of functions, the agent's capacity to effectively select and utilize the appropriate functions and arguments diminishes as the number of available functions increases. This is particularly evident in complex scenarios involving a multitude of potential actions and choices. Such an overload leads to drastic inaccuracies in function name selection and argument formulation. By contrast, our multi-agent architecture ensures that each agent has access only to a specific subset of functions relevant to their designated role. This targeted access significantly enhances the precision of function calling, as each agent specializes in a particular domain, ensuring more accurate and contextually appropriate function and argument selections. Another critical aspect is the management of context length in chat-based interactions. In a single-agent system tasked with handling an entire trip-planning conversation, the necessity to retain all context within the agent's limited memory poses a great challenge: As trip plans become more intricate, it becomes infeasible to fit the entire chat history within the model's context length. This limitation often leads to suboptimal results, as the agent might either forget previous parts of the conversation or make poor judgments due to incomplete context. Methods like context summarization, while useful, tend to be lossy and exacerbate these issues as the conversation extends. Our hierarchical multi-agent structure effectively addresses this challenge. In this setup, lower-level details are only retained within the context of subordinate agents, while higher-level agents in the hierarchy are not burdened with these granular detail: Instead, they receive summarized reports from the lower levels. This approach not only keeps the context length manageable, even in extended conversations but also optimizes the use of the model provider's API. Since we do not need to include every detail of the trip history in API calls, the queries become more focused and cost-effective. In summary, our decision to employ a hierarchical multi-agent system is a strategic choice to enhance scalability, function calling accuracy, and effective context length management. This architecture ensures that our system remains robust, efficient, and capable of handling complex trip-planning tasks with a high degree of precision and user satisfaction.

C. Group Chat Implementation

The group chat starts with the group chat manager coordinating a chat with the system prompt: "Hey everyone cooperate and help Agent Brain in his task of generating a good trip plan." When the user submits their query, the brain takes the user's query and initiates a chat with the group chat manager by passing in the query as the first message in the group chat. Hence, the groupchat manager starts this round-based chat and selects one agent to speak in each round. For this, we defined a custom speaker selection logic so that the group chat manager is allowed to select only the team leaders to speak, and team leaders themselves suggest their most relevant team members to speak by appending an uppercase "NEXT:" message at the end. Every round, the group chat manager makes a decision based on the suggested next speaker by the previous agent and the overall all conversation history to select the next speaker. Thus, teams only talk within themselves. After their team-internal discussion ends, team leaders report back to the group chat manager, and a new team leader is selected to speak, and so on. During this discussion, team members are allowed to call the low-level agents (websearch, booking.com API etc.) for "help" before reporting back. Except termination user proxy, no agent has to ability to terminate the group chat. After each team's discussion ends and the group chat manager is satisfied, the manager selects brain to speak which asks the user if they are satisfied with the answer. If they are, the group chat manager selects the termination user proxy to speak whose whole purpose is to terminate the conversation and return to the user.

```
    "You are team member TouristicAttractionFinder. You know about touristic
attractions and can provide "
            "information about them. You MUST only use the tools provided to you "
            "with inputs relating to tourist attractions and nothing else."
            " You MUST first use webSearch, after executing the webSearch, you must
call update_travel plan to update your section
(EventPlanning.TouristicAttractionFinder) with event in the format:"
            """       {
                "TouristicAttractions": {
                "Attraction1": {
                    "Name": "",
                    "Description": "",
                    "OpeningHours": "",
                    "EntryFee": ""
                    #// Additional attractions can be added in a similar format
                }
                }
            },"""
            "You MUST end your message with 'NEXT: HeadofEventPlanning' and report
back your results."
```

Figure 2: An Example System Prompt for A Single Agent, TouristicAttractionFinder

D. Shared Object

The entire group chat serves to populate a shared JSON object (so every agent has access to update, delete, and add information to this object). In the system prompts of team members, they are given the section they are responsible for updating in the shared object. For instance, TouristicAttractionFinder agent has the system prompt displayed in Fig. 2.

Each team member has similar system prompts. They have two designated functions; one for updating the shared object, and another for using other tools (such as web search, booking API, etc) When their team leaders suggest them to the speaker, in their turn, team members take the relevant information and call their low-level agent to access real-time information. Then, they call the update_travel_plan function. This function takes the section to be updated, and the data to

insert (a JSON object as well). Finally, the team member updates the shared object with this function, shares the findings in the group chat and reports back to the team leader. Since each agent has assigned sections in this object, our solution is similar to multiple people updating a shared Google Doc with different access rights. So our idea of collaborating to create a common final product is applicable to any domain that has similar problems.

III. Evaluation

Our evaluation uses an Alpaca Farm [2] style LLM judge for scaling and consistency with planning agent evaluations. Furthermore, since travel planning metrics have not been established in the research literature, we develop five categories for travel planning evaluation using a compilation of 20+ travel itineraries and checklists.

TABLE I

EVALUATION RESULTS USING OUR JUDGE LLM

Category	Human Travel Plan	TripChat Travel Plan
Destination Accuracy	99%	88%
Travel Event Personalization	99%	77%
Budget Management and Value Alignment	99%	88%
Logistical Coherence and Convenience	88%	55%
Cultural Engagement and Experiential Richness	99%	88%
Non-hallucinatory	100%	99%

The evaluation metrics were meticulously designed to address each aspect of travel planning. First, the Destination Accuracy metric ensures the selected location resonates with the traveler's interests, be it cultural immersion, adventure, or relaxation, forming the foundational element of the travel experience. Following this, Travel Event Personalization Alignment scrutinizes the itinerary's customization to the traveler's unique preferences, enhancing their overall experience by personalizing activities and events. In practicality, Budget Management and Value Alignment examines how the plan aligns with financial constraints while maximizing experience value, striking a crucial balance between affordability and quality. Logistical Coherence and Convenience then evaluate the arrangement's practicality, focusing on transportation, accommodation, and the logical sequencing of activities for a seamless and stress-free journey. Lastly, Cultural Engagement and Experiential Richness delves into the depth of cultural immersion and learning opportunities, enriching the traveler's connection with the destination. Collectively, these categories form a robust and holistic framework, ensuring each travel plan is comprehensively assessed from multiple critical dimensions. We have developed a comprehensive evaluation system for AI-generated travelplans by utilizing GPT-4 as the LLM judge across the five established categories. In order to compare the quality of our generated plans, we compare our plans to a set of 11 baseline plans manually curated by hand to maximize aspects of our 5 evaluation criteria. We then use the same LLM judge to evaluate our generated plans on the same metric, and then compare them. Initially, we encountered issues with the LLM being inconsistent with its evaluation on the manually curated baseline plans. However, we were able to make our judge much more consistent and accurate by using 2 different techniques for language model prompting: chain-of-thought prompting [3] and using in-context examples [1]. We used around 10 in context examples, with one positive and one negative example for each criteria. We used chain-of-thought for each in context example to allow the judge to understand why they should come to that conclusion. Our findings show that TripChat is almost as good as our manually curated human travel plan, with results shown in Fig. 3

IV. Conclusion

We show that TripChat can serve as a first step towards an automated trip planner, which can either be used by individuals or by travel agencies to save time and money. TripChat is novel because it demonstrates the ability to plan using real-world data, which also drastically reduces hallucinatory outputs. Moreover, the hierarchical multi-agent structure ensures that AI maintains a coherent and reliable line of reasoning throughout the planning process, aligning closely with the expert-level plans in our evaluations. As for future work, teams or agents whose tasks and results do not depend on each other can work in parallel to make the trip-planning process even faster and more efficient.

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhari-wal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 5

[2] Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Car los Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback, 2023. 4

[3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. 5

[4] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang (Eric) Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. Technical Report MSR- TR-2023-33, Microsoft, August 2023. 1