Implementing traffic agent based on LangGraph

Haitian Chen, Ye Ding*
Dongguan University of Technology, Dongguan, 523808, China
*dingye@dgut.edu.cn

ABSTRACT

As urbanization accelerates, the rapid increase in urban populations and vehicle numbers poses unprecedented challenges to city traffic. The demand for intelligent transportation systems, a key component of smart city construction, is growing day by day. This system is a highly integrated interdisciplinary field that combines complex computer algorithms, which to some extent limits its extensive application. The emergence of large language middleware has reduced the deployment barriers for related applications. This article proposes an innovative intelligent transportation system that combines a large language model (LLM) with Amap API through LangGraph, providing strong support for urban traffic. This system not only assists users in making travel decisions through natural language dialogue based on the superior reasoning and planning capabilities of LLM, but also enhances the semantic understanding ability of LLM by constructing a graph structure through LangGraph,it improves the capture capability for urban traffic tasks and implements a self-feedback mechanism for the large language model.This innovative approach offers a more convenient and efficient method for the application of large language models in the field of Intelligent Traffic System.

Keywords: Large Language Models, ITS, LangGraph, Amap

1. INTRODUCTION

Transportation has been an essential part of human history, and with the advancement of computer technology, using computers to assist humans in making optimizations and decisions in the field of intelligent transportation has become a revolutionary approach, providing a new direction for improving human life. In areas such as traffic simulation, route planning, and intelligent scheduling, researchers are continuously applying new technologies to enhance the efficiency and convenience of transportation systems.

The development of artificial intelligence (AI), especially its combination with natural language processing, has brought unprecedented opportunities to the field of intelligent transportation. Large language models like ChatGPT[1] are leading a technological revolution worldwide. Zheng et al.[2] explored the potential applications of large language models (LLMs), particularly ChatGPT, in intelligent transportation systems in their study. OpenAI[3] and TrafficGPT[4] both explored the viability of Large Language Models as a control agent that selects actions from the action space,to assist humans in making traffic control decisions through natural language dialogue.

This article introduces an innovative dialogue framework for smart city transportation attributes, built on the LangGraph middleware and achieving seamless integration with the Amap API interface. Through this integra- tion, we have provided powerful tools for large language models to clearly identify and solve specific problems.By using the LangGraph, we have integrated tools for LLM and constructed a network representing the agent system with Graph to help LLM understand and analyze problems more effectively. We propose an LLM agent for intelligent transportation based on LangGraph, primarily responsible for handling city map interaction tasks via the Amap API. In urban traffic tasks related to route planning, the current mainstream approach involves training Neural Network Models with a large amount of traffic data, requiring different network structures for different tasks and consuming substantial training resources. Compared to conventional intelligent transportation systems combined with LLMs, our use of the LangGraph structure offers greater scalability. It realizes the issues of agent loop calling and coordination between multiple agents, making it the simplest way to implement LLM applications today. In the following parts of this article, we will delve into the architectural design, implementation methods, and achievements of this framework, demonstrating how it efficiently combines large language models with the field of smart city transportation.

Fourth International Conference on Intelligent Traffic Systems and Smart City (ITSSC 2024),
edited by Hao Chen, Wei Shangguan, Proc. of SPIE Vol. 13422, 1342226
 · 0277-786X · doi: 10.1117/12.3050639

2. RELATED WORK

Since the release of ChatGPT, numerous new middleware projects for large models have emerged, aiming to simplify the process for developers to build AI applications based on large language models (LLMs). Among them, Langchain[5] provides interfaces for multi-model access, prompt encapsulation[6] and multi-data source loading, greatly simplifying the construction of AI applications. Langchain, with its adaptability to multiple models, ease of integration, and strong support for mainstream programming languages like Python, has quickly gained popularity among developers.

2.1 LangGraph

LangChain connects large models with external interfaces by constructing a chain structure. The LangChain Expression Language (LCEL) is a simple representation method provided by LangChain for assembling chains. Chains assembled in this way can automatically acquire a series of capabilities such as batch processing, streaming output, parallel processing, and asynchronous processing. Additionally, chains can be further assembled into more complex chains and agents through LCEL. However, chains constructed in LangChain do not have looping capabilities and cannot be called in a loop during inference.

Therefore, LangGraph is used to deal with such problem. It is not an independent framework from LangChain but an extension library built on top of LangChain. LangGraph can coordinate multiple chains, agents, and tools to work together to complete input tasks, and supports more refined control of looping and agent processes in LLM calls. The implementation of LangGraph involves using a new form, the StateGraph, to construct the previously black-boxed calling process based on AgentExecutor. By precisely defining the details of LLM-based tasks through graphs, the final application is compiled and generated based on this graph. During task execution, the state of graph is maintained and continuously updated based on node transitions.

The construction process of LangGraph includes defining nodes, edges, and state schema. It simplifies state management and interruption handling, enabling developers to focus on defining nodes, edges, and state architecture. StateGraph is the fundamental class in LangGraph that represents the entire state diagram. It stores the states of various variables during workflow execution, represents the message format for communication between nodes in the graph, and reflects the execution process and states of the graph. Once a node is executed, it updates the state of the graph. The state of LangGraph is updated during the execution of the graph and is shared with all edges and nodes. This state defines the core state object that is updated over time, receives operations and attribute definitions, and is updated by nodes; these state information are transmitted between each node. Each node updates this state information based on its own defined logic.

2.2 Amap API

Amap is China's leading provider of digital map content, traffic, and location service solutions. The open API interfaces provided by Amap offer a wide range of practical tools for handling maps and adding content to maps via various services. It provides geographic data service interfaces such as the geocoding or reverse geocoding API, route planning API and search API. These interfaces access remote services via internet protocols, enabling functionalities like the conversion between structured addresses and latitude/longitude, as well as the development of route planning features. These services can be used to integrate different applications, allowing developers to provide users with rich map-related functions such as navigation, route planning, and geographic location queries. The traffic agent returns information by making real-time calls to the Amap API, which uses real-time traffic status, ensuring that the returned information is reliable.

3. IMPLEMENT AMAP CALL ON LANGGRAPH

3.1 Tool construction

The tools we defined are constructed based on the API interfaces provided by Amap. We have encapsulated functions for city area encoding, weather queries, latitude and longitude queries, area positioning, walking route planning, bus route planning, and car route planning. Additionally, we integrated the Baidu search tool into our system through an app created using Baidu's APPbuilder, enabling us call the Baidu search tool.Following the same execution flow, there are different augmented tools that could help users with various requirements as presented in Table 1.

We need to follow certain requirements when building tools.We must ensure the effectiveness of the API interfaces to prevent errors caused by failing to retrieve relevant information due to interface failures. When defining a tool, we need to use the tool decorator to indicate the creation of the tool's invocation function. The comments preceding the function statements describe the tool. These descriptions are provided as prompts to the large model when binding the tool, enabling

the model to understand the tool's functionality. Therefore, when defining prompts, it is important to consider that the output of one tool might be the input for another during the model's reasoning process. The prompts should be written consciously and in a standardized manner to facilitate the model's inference and tool invocation. Listing 1 illustrates an example of building a utility function.

Listing 1. The structure of the geo tool.

```
@tool
def geo(city:str,add:str)-> str:
    """Obtain the geographic code of the specific location of a given city, which
       corresponds to latitude and longitude. The city parameter is the city
       name, and the add parameter is the specific location name of the city.
       The returned message contains latitude and longitude, with longitude at
       the beginning and latitude at the end"""
    url = 'https://restapi.amap.com/v3/geocode/geo?parameters'key
    private = 'API-KEY'
    param ={
        'key':key priva
        te,'address':add,
        'city':city,
        'output':'json'
```

Table 1. A list of augmented tools.

Augmentation Name	Description
geoGet	Used to obtain the latitude and longitude of a city
weatherGet	Obtain weather information through urban area coding
DistrictCoding	Obtain urban area coding
walkRoute	Pedestrian route planning
driveRoute	Driving route planning
busRoute	Bus travel planning
staticMapget	Obtain a map labeled with the corresponding location name
busRouteMapget	Obtain the queried bus route map
baidu search	Implementing Baidu Search Function through Baidu App Builder

3.2 LangGraph structure construction

The graph built with langgraph is shown in Figure 1. During the process of building structure the LangGraph, the messages between nodes are typically formatted as a list of basemessage. The data returned after each node's execution is encapsulated into a base message and wiil be appended to the message list as the latest entry.

The llm agent node is an agent node that calls the LLM bound to the above tool. LLM determines whether to call tool based on system prompt and user messages. If necessary, it returns an Aimessage that includes the input parameters, tool name, and tool ID. Message will pass the message to the action node, which is the ToolNode, through the edge, and the tool calls will be executed based on the tool ID, such as generating coordinates through the geoGet tool. After the tool executes and returns ToolMessage, which appends to state, the state message continues to return to the llm agent node. Llmaagent decides whether to complete the task or continue calling the tool until the task is completed.

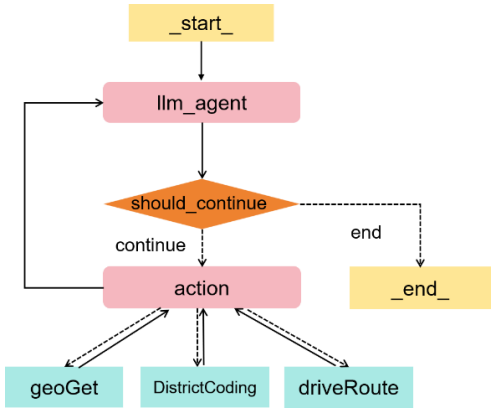

Figure 1. Execution flowchart after receiving a natural language task.

4. EXPERIMENT

In order to demonstrate the framework's efficiency in handling various complex transportation tasks, we have provided an example that utilizes ChatGPT (gpt-3.5-turbo) as the language model (LLM) and employs LangGraph to direct the operation of the LLM.

As an AI traffic assistant, it needs to engage in natural language conversations with users, execute basic tasks based on user instructions, and support multi-turn dialogues.

In the experiment, we will use multiple different large language models to compare their understanding and parsing abilities when handling relevant issues in our system. The selected models include gpt-3.5-turbo, glm-4, moonshot, and qwen-max, each of which has advantages in multi-language support, contextual understanding, long text processing, and multimodal interaction. We categorize the problems based on the complexity of user input into two types: single-tool call problems and multi-tool call problems. Single-tool call problems require the large language model to parse the user's information once, and call the appropriate tool to complete the task. Multi-tool call problems require the model to parse and call multiple tools as needed to complete a series of consecutive subtasks. In the experiment, we conducted more than 20 times with each large language model for each type of tool-specific problem.

The experimental results are shown in Figure 2. In the figure, we present evidence using three different evaluation dimensions displayed on a bar chart. The x-axis represents the large language models used, with the two types of tool invocation problems compared side by side. Based on the y-axis, we can compare the percentage values of correct results returned by each model.

We can observe that gpt-3.5-turbo, glm-4, and qwen-max demonstrate similar levels of understanding and parsing ability when addressing problems that require only a single tool invocation. However, when dealing with problems that require multiple tool invocations, the differences between these models become more pronounced.

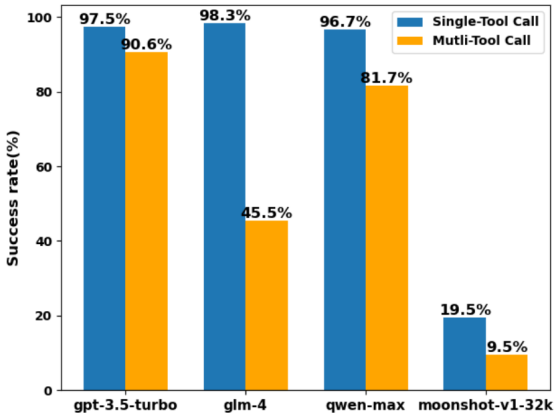

Figure 2. Comparison of lateral performance of different large models.

During the experiment, we found that gpt-3.5-turbo performed the best in handling multi-tool invocation problems. It calls tools one by one based on each specific need, rather than calling all potentially required tools at once. Moonshot, on the other hand, tends to call multiple tools simultaneously instead of sequentially. In contrast, gpt-3.5-turbo's ability to flexibly call tools based on actual needs improves its problem- solving efficiency. Qwen-max has some issues with tool invocation because it does not automatically include the tool id, re- quiring appropriate adaptations in the code to return correct results. Although glm-4 has good comprehension capabilities, it requires very clear system prompts and tool descriptions; otherwise, ambiguities can arise, pre- venting task completion. Overall, domestic models exhibit weaker understanding and generalization capabilities in multi-tool invocation tasks, resulting in lower success rates.

In comparison with other traffic-related methods, we selected Open-Ti[3] and TrafficGPT.[4] Under the same conditions, using ChatGPT as the base large language model, we compared the accuracy of correct responses to relevant questions within their respective domains across the three methods. As shown in the Figure 3, our method significantly outperforms the baseline methods in terms of accuracy.

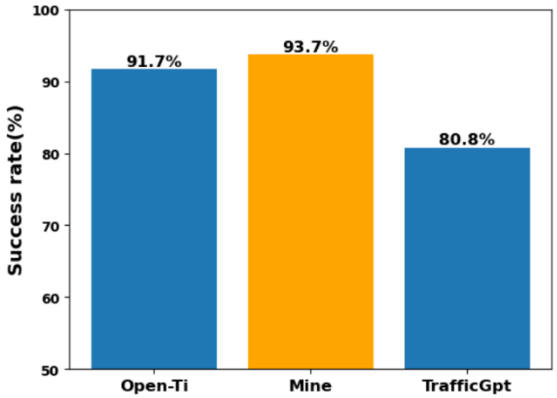

Figure 3. Comparison with Open-Ti and TrafficGPT in terms of success rate.

5. CONCLUSION

In summary, this study implemented a tool based on LangGraph that combines Amap's API with large language model, integrating the contextual understanding capabilities of the language model with traffic-related tools to enhance its functionality. This combination allows for question answering and solutions related to smart city traffic. By incorporating Amap's API, the tool improves the large language model's ability to analyze specific problems, ensuring the reliability and accuracy of its responses. Additionally, it facilitates the intelligent deconstruction of complex tasks, leveraging the parsing capabilities of the large language model to gradually complete these tasks. Through natural language dialogue, this tool plays a key role in assisting humans in making traffic decisions.

During the experiments, we found that due to some compatibility issues, the parsing capabilities of different large language models were not fully realized when relying on the LangGraph structure. However, this is not due to the model's inherent limitations but rather a temporary technical issue. In this agent, only some basic functions are currently bound, and more complex functions have not yet been implemented. Additionally, due to reliance on Amap's API, some of the interfaces provided by Amap have traffic limitations and cannot be called continuously. In the future, functionalities can be implemented by locally deploying a large model, providing data through Retrieval-Augmented Generation (RAG) methods, or fine-tuning the large model. Moreover, due to LangGraph's strong extensibility, multiple agents can be used to achieve different functions independently and collaborate with each other, further enhancing its problem-solving capabilities.

REFERENCES

[1] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G., "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys* **55**(9), 1–35 (2023).
[2] Zheng, O., Abdel-Aty, M., Wang, D., Wang, Z., and Ding, S., "Chatgpt is on the horizon: Could a large language model be suitable for intelligent traffic safety research and applications?," (2023).

[3] Da, L., Liou, K., Chen, T., Zhou, X., Luo, X., Yang, Y., and Wei, H., "Open-ti: Open traffic intelligence with augmented language model," *International Journal of Machine Learning and Cybernetics* , 1–26 (2024).
[4] Zhang, S., Fu, D., Liang, W., Zhang, Z., Yu, B., Cai, P., and Yao, B., "Trafficgpt: Viewing, processing and interacting with traffic foundation models," *Transport Policy* **150**, 95–105 (2024).
[5] Topsakal, O. and Akinci, T. C., "Creating large language model applications utilizing langchain: A primer on developing llm apps fast," in [*International Conference on Applied Engineering and Natural Sciences*], **1**(1), 1050–1056 (2023).
[6] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C., "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382* (2023).