In-Vehicle Artificial Intelligence Agent System with Local Large Language Models for Agent Selection and Vehicle Control

Toru Furusawa*
*Toyota Motor Corporation, Tokyo, Japan

***Abstract*—Voice User Interfaces (VUIs) in vehicles enhance driver experience by enabling hands-free interaction with in-vehicle systems. However, the increasing complexity of these systems poses challenges for intuitive use, as drivers must remember numerous voice commands. Recent advancements in large language models (LLMs) offer opportunities to create more natural and flexible voice interactions. In this paper, we propose an in-vehicle Artificial Intelligence (AI) agent system that utilizes a local LLM for agent selection and vehicle control, eliminating the need for manual agent selection and dependency on internet connectivity for vehicle Application Programming Interface (API) execution. The system analyzes user speech content to automatically select appropriate AI agents and executes in-vehicle APIs using a local LLM capable of function calling. We implemented the system on an NVIDIA Jetson AGX Orin Developer Kit to simulate in-vehicle devices and evaluated its performance in terms of response time, accuracy of in-vehicle control API execution, and function calling success rate. Experimental results demonstrate that the proposed system achieves in-vehicle API execution with acceptable latency, improving usability and safety by enabling seamless and natural voice interactions without relying on internet connectivity.**

***Index Terms*—Large language models, in-vehicle AI agents, edge computing, function calling**

I. Introduction

A. Background

Voice User Interfaces (VUIs) that facilitate dialogues with Artificial Intelligence (AI) agents significantly enhance the user experience for vehicle drivers, who need to keep their hands on the wheel at all times [1]. Various Original Equipment Manufacturers (OEMs) have developed numerous in-vehicle VUI applications for tasks such as phone calls, music playback, navigation, and vehicle control.

However, as the complexity of in-vehicle systems increases, intuitive use of VUI applications by drivers becomes more challenging. Current VUIs often rely on pre-defined voice commands for task execution, making it cumbersome for drivers to remember commands for every application. Enabling more flexible voice dialogues would allow drivers to use in-vehicle system features through natural conversations, thereby enhancing their driving experience.

The advancement of Large Language Models (LLMs) has sparked considerable interest in voice-interactive AI agents [2], with conversational AI agents and chatbots being among the most utilized applications. The potential for voice interactions with a variety of AI agents is vast. These interactions can range from simple conversations to providing educational advice and tourism information. They can also include vehicle consultations and even interactions with digital avatars of oneself or family members. This variety promises to meet the diverse needs of drivers.

B. Previous Work and Challenges

In our previous work [3], we proposed a voice-interactive AI agent system for connected vehicles that employs multiple LLMs distributed across in-vehicle devices, edge servers, and public cloud services. This system allows drivers to select manually from various AI agents based on their specific needs, with the deployment of LLMs optimized according to the characteristics of each agent.

While this approach addressed some challenges of integrating LLMs into in-vehicle systems, it had two significant limitations:

1) **Manual AI Agent Selection:** When multiple AI agents were available, users had to explicitly select the AI agent to use. For drivers, manually switching AI agents during conversations is cumbersome and detracts from the user experience. Additionally, using Graphical User Interfaces (GUIs) for selecting could interfere with driving and pose safety risks.
2) **Dependency on Internet Connectivity for Vehicle APIs:** The system utilized Function Calling [1], a cloud service provided by OpenAI, for dynamic selection and execution of in-vehicle APIs. This reliance on Function Calling required an internet connection for executing local in-vehicle APIs. This dependency rendered the system unusable in areas without internet connectivity. Given that in-vehicle API selection and execution can affect usability and safety, it is desirable for the vehicle control agent to operate anywhere, regardless of internet connectivity.

C. Objective and Contribution

To address these challenges, we propose a new system architecture that enables the selection of AI agents based on the user's speech content and the selection and execution of vehicle APIs using a local LLM running on in-vehicle

[1]https://platform.openai.com/docs/guides/function-calling

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

devices. By performing these functions locally, the system offers several advantages, including independence from internet connectivity and enhanced security.

The contributions of this study are as follows:

- **Development of Distribution Agent:** We propose a method to automatically select AI agents based on speech content, eliminating the need for manual selection or reliance on GUIs. This significantly enhances the usability and safety of in-vehicle AI systems.
- **Development of Vehicle Control Agent:** We propose a method to execute in-vehicle APIs using a local LLM to replicate the functionality of Function Calling within the vehicle, enabling in-vehicle API operations without requiring internet connectivity.
- **Performance Evaluation in a Device-Edge Environment:** We conducted experiments in a simulated in-vehicle environment, demonstrating that the proposed system achieves AI agent selection and in-vehicle API execution within approximately 3.16 seconds. While slightly slower than the method with OpenAI API's Function Calling (1.99 seconds), subjective quality assessments showed no significant differences.
- **Enhancement of System Independence and Security:** By eliminating dependency on cloud services for critical operations, the system ensures functionality in areas with poor connectivity and improves security by keeping sensitive vehicle data local.

This study paves the way for more robust and user-friendly in-vehicle AI systems, providing drivers with a seamless and secure interaction experience, regardless of internet connectivity.

II. Proposed Method

A. System Overview

In this study, we propose an in-vehicle AI agent system that uses a local LLM to perform AI agent selection and in-vehicle API execution. The system overview is shown in Figure 1.

The main components of the system are as follows:

- **User Interface**: Receives user inputs and returns responses.
- **Distribution Agent**: Performs AI agents selection.
- **Vehicle Control Agent**: Selects and Executes in-vehicle APIs.
- **Cloud AI Agents**: Perform various tasks such as chat, music recommendations, and tour guides on the cloud.

B. Distribution Agent

Distribution Agent manages the state of all other AI agents and distributes user inputs to the appropriate AI agent based on the content of the user's speech by performing function calling with the local LLM. It also directly processes simple responses such as greetings without relying on cloud AI agents, thereby reducing external communication.

The system sequence of AI agent selection by Distribution Agent is shown in Figure 2. The sequence descriptions are as follows:

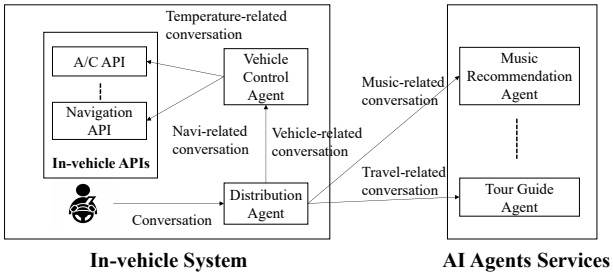


Fig. 1. System overview

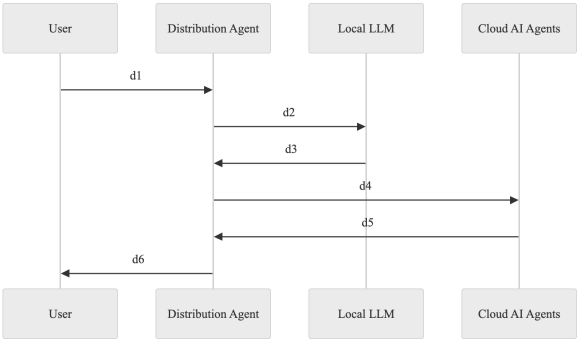


Fig. 2. System Sequence of AI Agent Selection

- **d1**: The user sends user input to Distribution Agent.
- **d2**: Distribution Agent sends user input and a set of descriptions of available AI agents to the local LLM.
- **d3**: The local LLM provides the API endpoint of the selected AI agent which is considered to be most relevant to the content of the user input to Distribution Agent.
- **d4**: Distribution Agent sends user input to the selected AI agent.
- **d5**: The selected AI agent sends the response to Distribution Agent.
- **d6**: Distribution Agent sends the response by the selected AI agent to the user.

C. Vehicle Control Agent

Vehicle Control Agent performs function calling with the local LLM to execute in-vehicle APIs to interpret natural language commands from the user and execute appropriate vehicle control functions. The in-vehicle APIs operate on the in-vehicle device, enabling direct control.

The system sequence of in-vehicle API execution including the AI agent selection is shown in Figure 3. The sequence descriptions are as follows:

- **v1**: The user sends user input to Distribution Agent.
- **v2**: Distribution Agent sends user input and a set of descriptions of available AI agents to the local LLM.

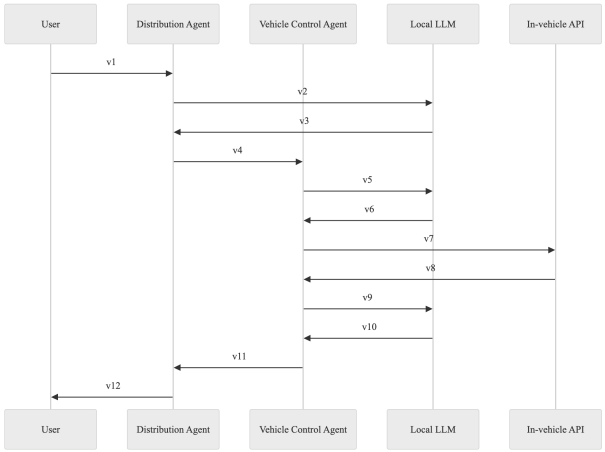


Fig. 3. System Sequence of In-vehicle API Execution

- **v3**: The local LLM provides the API endpoint of Vehicle Control Agent which is considered to be most relevant to the content of the user input.
- **v4**: Distribution Agent sends user input to Vehicle Control Agent.
- **v5**: Vehicle Control Agent sends user input and a set of descriptions of available in-vehicle APIs to the local LLM.
- **v6**: The local LLM provides the API endpoint of the selected in-Vehicle API and arguments which is considered to be most relevant to the content of the user input.
- **v7**: Vehicle Control Agent sends the arguments and execute the selected in-vehicle API.
- **v8**: In-vehicle API sends the response by the selected in-vehicle API.
- **v9**: Vehicle Control Agent sends user input and the response by the in-vehicle API execution to the local LLM.
- **v10**: The local LLM provides the response message to tell the user the result of the in-vehicle API execution to Vehicle Control Agent.
- **v11**: Vehicle Control Agent sends the response message to Distribution Agent.
- **v12**: Distribution Agent sends the response message to the user.

III. Implementation

A. Hardware Configuration

Given that future in-vehicle devices are expected to use edge devices equipped with Graphical Processing Units (GPUs), we use the NVIDIA Jetson AGX Orin Developer Kit to emulate the computational environment of an in-vehicle device. The main hardware specifications are shown in Table I.

B. Local LLM (Functionary)

To implement function calling on the local device, we adopt Functionary [2], an open-source local LLM that supports function calling and is released under the MIT license. Functionary is selected because it is the highest-performing local LLM among those listed on the Berkeley Function-Calling Leaderboard [3] at the time of our investigation.

IV. Evaluation

We evaluate the proposed system in the implementation environment. When a user inputs speech related to the in-vehicle API, we confirm that the Vehicle Control Agent is selected based on the content of the speech, and further, that the in-vehicle API related to the speech content is selected and executed. We also measure the response time from receiving the user input to returning the response message.

For comparison, we also show the results when using the OpenAI API (gpt-4o) instead of the local LLM Functionary.

A. AI Agents Services for Evaluation

We prepare two AI agent services to run on the cloud:

- **Music Recommendation Agent**: An agent that recommends songs on a music streaming service based on user preferences and requests.
- **Tour Guide Agent**: An agent that provides information on local tourist attractions.

In addition to these two agents, the Vehicle Control Agent operates on the in-vehicle device, so there are three AI agents to choose from for the Distribution Agent.

B. In-vehicle APIs for Evaluation

We prepare a total of 14 in-vehicle APIs in three types as shown in Table II. The first type of API is the Window Control API, which controls the opening and closing of the vehicle's windows. APIs are provided for the windows of the four seats. The second type of API is the Air Conditioner Settings API, which controls the air conditioner settings. APIs are provided to turn the air conditioner on/off and set the temperature. The third type of API is the Auto Cruise Settings API, which controls the auto cruise settings. APIs are provided to start/stop the auto cruise and set the speed.

All APIs are dummy APIs that do not actually control the vehicle's functions, but simply receive input and output responses.

TABLE I
Hardware Specifications

Specification	Details
AI Performance	Up to 275 TOPS (INT8)
GPU	NVIDIA Ampere architecture, 2048 CUDA cores and 64 Tensor cores
CPU	12-core Arm Cortex-A78AE v8.2 64-bit CPU (3MB L2 + 6MB L3)
Memory	64GB 256-bit LPDDR5 (204.8 GB/s)

[2]https://github.com/MeetKai/functionary

[3]https://gorilla.cs.berkeley.edu/leaderboard.html

C. Evaluation Scenarios

For the evaluation, we define 15 scenarios that involve calling the in-vehicle APIs as shown in Table III. We sequentially execute the 15 scenarios, measure the response time including the processing time for some internal sequences, and verify that the Vehicle Control Agent is selected, and the expected in-vehicle API is executed, as well as the accuracy of the expected response content.

D. Evaluation Results

For both the local LLM (Functionary) and the OpenAI API (gpt-4o) function calling, we confirmed that the expected in-vehicle API was executed and the correct response was returned for all scenarios.

The evaluation results for response times are shown below. Figure 4 summarizes the response time taken to execute the in-vehicle APIs for each scenario using both the local LLM and the OpenAI API for function calling. Table IV shows the average response time for each sequence in the 15 scenarios.

It also shows the internal processing time between some sequences in Figure 3. $v1 - v3$ indicates the time from when the user sends the input to when the Distribution Agent queries the local LLM and selects the Vehicle Control Agent. $v4 - v6$ indicates the time from when the Distribution Agent sends the user input to the Vehicle Control Agent and queries the local LLM to select the appropriate in-vehicle API. $v7 - v8$ indicates the time from when the Vehicle Control Agent executes the in-vehicle API to when it obtains the result. $v9 - v12$ indicates the time from when the Vehicle Control Agent requests message generation from the local LLM to when the generated message is returned to the user. These classifications are made to clearly show the processing time for each sequence, as queries to the local LLM or OpenAI API occur in each sequence. The total time for $v1 - v3$, $v4 - v6$, $v7 - v8$, and $v9 - v12$ is the time from when the user sends the input to when the response is returned.

TABLE II
LIST OF VEHICLE APIS USED FOR EVALUATION

Window Control APIs
open_front_left_window()
close_front_left_window()
open_front_right_window()
close_front_right_window()
open_rear_left_window()
close_rear_left_window()
open_rear_right_window()
close_rear_right_window()
Air Conditioner Settings APIs
turn_on_air_conditioner()
turn_off_air_conditioner()
set_air_conditioner_temperature(temp)
Auto Cruise Settings APIs
enable_auto_cruise()
disable_auto_cruise()
set_auto_cruise_speed(speed)

First, we compare the total response times. The average total response time for the 15 scenarios using the local LLM is 3.16 seconds, while the average for the OpenAI API is 1.99 seconds. While the local LLM does not incur communication delays due to the lack of external communication, the longer execution time of the local LLM on the local device compared to the Open AI API is the cause of the difference. However, this difference is not considered to have a significant impact on the user experience.

Next, we examine the response time in the $v1 - v3$ sequence interval. When using the local LLM, the average response time in this interval was consistently around 1.41 seconds. This indicates that the processing time for AI agent selection in the Distribution Agent was consistent, as the number of candidate AI agents was small and constant at 3 agents. In contrast, when using the OpenAI API, the average response time in this interval was 0.66 seconds, which is shorter than when using the local LLM, but with some variation. In addition to communication delays, the processing time of the OpenAI API itself is likely causing the variation.

Regarding the response time in the $v4 - v6$ sequence interval, there is variation in the processing time even when using the local LLM. This is likely due to the large number of candidate in-vehicle APIs, which is 15.

For the response time in the $v9 - v12$ sequence interval, the processing time was consistently around 0.40 seconds when using the local LLM. This indicates that the processing time is short and consistent for simple message generation. When using the Open AI API, the average response time is 0.59 seconds, which is the only interval where the processing time is longer than when using the local LLM. This suggests that the local LLM on the device may have superior response times not only for AI agent selection and in-vehicle API selection, but also for simple response message generation compared to using external cloud services.

V. DISCUSSION

A. Analysis of Results

The evaluation results demonstrate that the proposed system successfully performs in-vehicle API execution using a local LLM with acceptable latency. While the local LLM exhibits slightly higher response times compared to the cloud-based OpenAI API, the difference of approximately 1.17 seconds does not significantly impact the user experience. The system consistently executed the expected in-vehicle APIs and provided correct responses across all evaluated scenarios. These findings indicate that the local LLM is capable of handling function calling and agent selecting effectively in an in-vehicle environment, fulfilling the objectives of reducing dependency on internet connectivity and enhancing responsiveness.

B. Comparison with Cloud-based Systems

Compared to cloud-based systems, the proposed architecture offers several advantages. By processing AI agent selection

TABLE III
EVALUATION SCENARIOS FOR VEHICLE API FUNCTION CALLING

Scenario	User Input	Expected Vehicle API	Expected Response
1	What's the air conditioner temperature setting value?	get_air_conditioner_temperature()	"The air conditioner temperature is set to 22.0 ° C."
2	Please set the air conditioner temperature to 20.0.	set_air_conditioner_temperature(20.0)	"The air conditioner temperature has been set to 20.0 ° C."
3	What's the current air conditioner temperature setting value?	get_air_conditioner_temperature()	"The air conditioner temperature is set to 22.0 ° C."
4	Hey, change the air conditioner temperature to 24.0.	set_air_conditioner_temperature(24.0)	"The air conditioner temperature has been set to 24.0 ° C."
5	What's the current air conditioner temperature?	get_air_conditioner_temperature()	"The air conditioner temperature is set to 22.0 ° C."
6	Tell me which windows are open.	get_open_windows()	"The driver's window is open."
7	Please open the driver's window.	open_front_left_window()	"The driver's window has been opened."
8	Please open the passenger's window.	open_front_right_window()	"The passenger's window has been opened."
9	Tell me which windows are open.	get_open_windows()	"The driver's window and the passenger's window are open."
10	Start autocruise with 60 km/h.	enable_auto_cruise() set_auto_cruise_speed(60)	"Autocruise has been started at 60 km/h."
11	How fast is the autocruise?	get_auto_cruise_speed()	"The current autocruise speed is 60 km/h."
12	Set the autocruise speed to 70 km/h.	set_auto_cruise_speed(70)	"The autocruise speed has been set to 70 km/h."
13	How fast is the autocruise?	get_auto_cruise_speed()	"The current autocruise speed is 70 km/h."
14	Stop autocruise.	disable_auto_cruise()	"Autocruise has been stopped."
15	How fast is the autocruise?	get_auto_cruise_speed()	"Autocruise is currently off."

TABLE IV
AVERAGE RESPONSE TIME TO EXECUTE IN-VEHICLE API

Sequence Term	Functionary	GPT-4o
$v1 - v3$	1.41 s	0.66 s
$v4 - v6$	1.29 s	0.69 s
$v7 - v8$	0.05 s	0.05 s
$v9 - v12$	0.40 s	0.59 s
Total	3.16 s	1.99 s

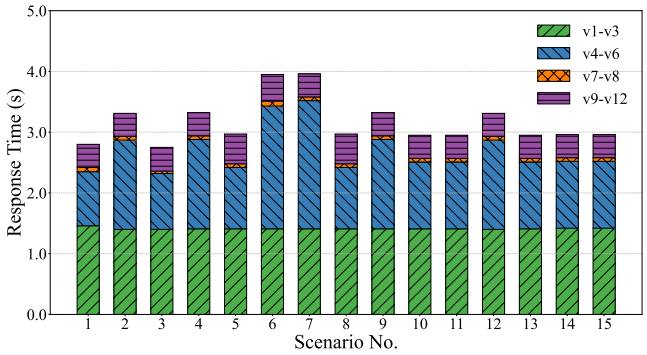
and in-vehicle API execution locally, the system operates independently of internet connectivity, ensuring consistent functionality even in areas with poor or no network coverage. Furthermore, local processing enhances security and privacy by keeping sensitive vehicle operation data within the vehicle. However, cloud-based systems may offer superior performance in handling complex dialogues requiring extensive reasoning or access to up-to-date external information. Therefore, the proposed system strikes a balance by delegating simple tasks to the local LLM and more complex tasks to Cloud AI agents.

C. Privacy Considerations

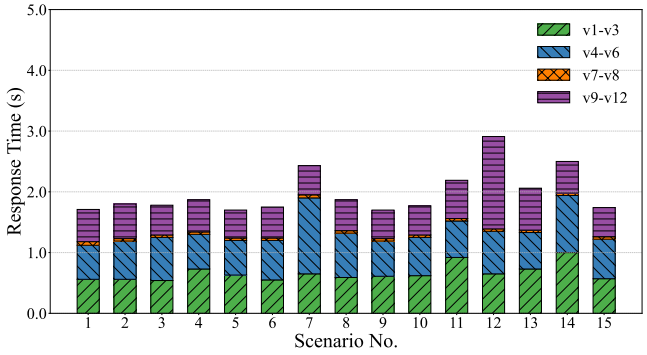
By processing simple responses and vehicle control commands locally, the system minimizes the transmission of sensitive user data to external servers. This enhancement in privacy protection is crucial in an era where data security concerns are paramount. Users can interact with the AI agent with greater confidence that their personal information and driving behaviors are not being transmitted or stored externally without necessity.

D. Limitations

Despite its advantages, the proposed system has certain limitations. The increased latency observed with the local LLM, although acceptable, may affect the user experience in scenarios requiring immediate responses. The computational resources required to run the local LLM, such as processing power and memory, may not be readily available in all in-vehicle devices, potentially increasing costs. Additionally, the capabilities of the local LLM may be limited compared to state-of-the-art cloud-based models, potentially affecting the complexity of tasks it can handle locally. There is also a



(a) Local LLM



(b) OpenAI API

Fig. 4. Segmented Response Time to Execute In-vehicle API

need to manage and update the local LLM to ensure it remains effective over time, which may introduce maintenance challenges.

E. Future Directions

Future work can focus on optimizing the performance of the local LLM to reduce latency and resource consumption, possibly through model compression techniques or hardware acceleration. Exploring more advanced local models or hybrid approaches that intelligently distribute tasks between local and cloud resources could further enhance performance and capabilities. Integration with more sophisticated AI agents and expanding the range of controllable vehicle functions can improve the system's versatility.

Additionally, implementing robust security measures to protect the system from potential threats and ensuring compliance with privacy regulations will be important considerations in future developments.

VI. Related Work

Research on systems using LLMs for vehicles is active, particularly in the application of multimodal LLMs for autonomous driving [4]. In [5], a method for commanding autonomous vehicles using natural language voice commands is proposed. In [6], a method for using LLMs in the decision-making process of autonomous driving systems is proposed.

Research on Function Calling, which dynamically executes necessary functions and APIs based on content, is also active [7]. In [8], a method for realizing Function Calling on resource-constrained edge devices is proposed. However, there is little research on executing in-vehicle APIs and switching agents using Function Calling in a voice-interactive AI agent environment within vehicles.

VII. Conclusion and Future Work

In this paper, we proposed an in-vehicle AI agent system that leverages a local LLM for agent switching and in-vehicle API control, addressing the challenges of manual agent selection and dependency on internet connectivity. By implementing function calling on a local LLM, the system can analyze user speech content to automatically select appropriate AI agents and execute vehicle control commands without relying on external services. Experimental evaluations demonstrated that the system achieves acceptable latency and accurate execution of in-vehicle APIs, enhancing usability and safety by enabling seamless and natural voice interactions. While the local LLM introduces slightly higher response times compared to cloud-based models, the benefits of improved privacy, security, and independence from network connectivity make it a promising approach for in-vehicle applications. Future work will focus on optimizing the local LLM's performance, expanding its capabilities, and exploring hybrid architectures to further improve the system's efficiency and user experience.

References

[1] D.-h. Kim and H. Lee, "Effects of user experience on user resistance to change to the voice user interface of an in-vehicle infotainment system: Implications for platform and standards competition," *International Journal of Information Management*, vol. 36, no. 4, pp. 653–667, 2016.

[2] O. Topsakal and E. Topsakal, "Framework for a foreign language teaching software for children utilizing ar, voicebots and chatgpt (large language models)," *The Journal of Cognitive Systems*, vol. 7, no. 2, pp. 33–38, 2022.

[3] T. Furusawa and M. Saitoh, "A demonstration of voice-interactive ai agents for vehicles utilizing multiple llms," in *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*. Los Alamitos, CA, USA: IEEE Computer Society, Jul 2024, pp. 249–251. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SMARTCOMP61445.2024.00058

[4] C. Cui, Y. Ma, X. Cao, W. Ye, Y. Zhou, K. Liang, J. Chen, J. Lu, Z. Yang, K.-D. Liao *et al.*, "A survey on multimodal large language models for autonomous driving," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 958–979.

[5] Y. Ma, C. Cui, X. Cao, W. Ye, P. Liu, J. Lu, A. Abdelraouf, R. Gupta, K. Han, A. Bera *et al.*, "Lampilot: An open benchmark dataset for autonomous driving with language model programs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 141–15 151.

[6] C. Cui, Y. Ma, X. Cao, W. Ye, and Z. Wang, "Receive, reason, and react: Drive as you say, with large language models in autonomous vehicles," *IEEE Intelligent Transportation Systems Magazine*, vol. 16, no. 4, pp. 81–94, 2024.

[7] W. Liu, X. Huang, X. Zeng, X. Hao, S. Yu, D. Li, S. Wang, W. Gan, Z. Liu, Y. Yu *et al.*, "Toolace: Winning the points of llm function calling," *arXiv preprint arXiv:2409.00920*, 2024.

[8] L. E. Erdogan, N. Lee, S. Jha, S. Kim, R. Tabrizi, S. Moon, C. Hooper, G. Anumanchipalli, K. Keutzer, and A. Gholami, "Tinyagent: Function calling at the edge," *arXiv preprint arXiv:2409.00608*, 2024.