Group 1 Final Report: Exploring Local LLM-based Copilot Capabilities for Enhanced Vehicle Functionalities

Yuhang Qian yuhangqi@usc.edu **Enyu Zhao** enyuzhao@usc.edu **Charlene Yuen** yuenchar@usc.edu **Kevin Zheng** zhengkev@usc.edu **Tianyu Bai** tbai@usc.edu

1 Abstract

This study aims to enhance vehicle functionality by deploying LLM (Large Language Model)-based copilots on local GPUs. While advanced vehicle voice assistants like Alexa struggle with complex multi-step commands, GPUs in electric vehicles present an opportunity for API call generation using locally loaded LLMs. We explore the effectiveness of using local LLMs to understand and execute complex commands directly on these GPUs to avoid the latency, costs, and internet reliance of cloud-based solutions. This approach may significantly enhance vehicle functionality by leveraging the impressive text-to-code and task decomposition capabilities of modern LLMs.

2 Introduction

Recent advancements in Large Language Models (LLMs) demonstrate considerable potential to revolutionize human-machine interfaces. With the impressive text-to-code and task decomposition capabilities many LLMs possess today, it is intriguing that advanced vehicle voice assistants such as Siri and Alexa still struggle with complex, multi-step commands. GPUs for autonomous driving are often found in electricle vehicles (EVs), but they remain largely underutilized during normal driving conditions. This research aims to explore the feasibility of deploying LLM-based agentic vehicle copilots on local GPUs to understand and execute complex commands. Ideally, this local approach will enhance vehicle functionality while minimizing the cost and latency associated with alternative cloud-based LLM approach.

3 Related work

3.1 Large Language Models

Recent rapid advancements in LLMs such as OpenAI's GPT series, Google's Gemini, and Claude Opus (OpenAI, 2021, 2024; Google, 2024; Anthropic, 2024) have transformed natural language processing. Despite their incredible capabilities, these models are often resource-intensive and closed-source. Smaller open-source LLMs like Llama 2-7B, 3-8B, and Mistral-7B (Touvron et al., 2023; Jiang et al., 2023) become promising alternatives for resource-constrained enviornments.

3.2 LLM Agents

LLMs are increasingly purposed as controllers for AI agents (Wang et al., 2023), demonstrating problem-solving, planning, and execution capabilities through techniques like Chain-of-Thought (Wei et al., 2023) and problem refinement (Xi et al., 2023). Such agents have been seen to play roles as a news commentator (Tseng et al., 2023), medical assistant (Biswas et al., 2023), and personalized house cleaner (Wu et al., 2023), even aligning with social norms (Li et al., 2024). In interactive environments, they excel at few-shot task decomposition (Song et al., 2023) while long-term memory enhances their conversational skills (Zhong et al., 2023). Fine-tuning further boosts agent performance for specific tasks (Zeng et al., 2023). Gorilla is a series of models (Patil et al., 2023) specifically designed for generating function and API calls. Gorilla OpenFunctions-v2 is surpassed on this task only by larger models like GPT-4 variants and Claude 3. We attempt to use Gorilla as a benchmark due to the similar nature of its purpose to our task.

3.3 Low-Rank Adaptation (LoRA)

Traditional fine-tuning of LLMs is computationally demanding. Low-rank adaptation reduces resource requirements, introducing the option of fine-tuning on edge devices (Hu et al., 2021). Advancements like QLoRA and LoRA+ further optimize memory usage and training speed, enabling performance gains (Dettmers et al., 2023; Hayou et al., 2024).

4 Problem Description

Current vehicle voice assistants struggle with complex, multi-step commands. This is surprising given that modern EVs offer unique advantages for advanced language processing: **Resource Availability:** EVs have powerful, often underutilized GPUs for autonomous driving, offering potential cost and latency benefits over cloud-based solutions. **Communication Infrastructure:** Ethernet-based architectures and API-based protocols in EVs align well with the text-to-code capabilities of LLMs, enabling potentially seamless interaction with vehicle functions.

However, creating effective local LLM-based vehicle copilots involves several challenges: **Domain-specific Dataset Creation:** Lack of publicly available datasets for training and benchmarking LLMs on vehicle-specific workflows, complicated by proprietary automakers' APIs. **Complex Task Decomposition:** Developing techniques for LLMs to decompose complex user commands into executable sequences of API calls. **Model Selection Constraints:** Optimizing LLMs to operate within the memory limits of typical vehicle GPUs (16-24GB of VRAM). **Model Performance Benchmarking:** Fine-tuning LLMs for vehicle-specific datasets and rigorously comparing results against cloud-based LLM services to evaluate the local LLM approach.

This research aims to address these challenges and explore the potential for significantly enhancing the user experience of vehicle voice assistants using locally deployed LLM-based copilots.

5 Methods

5.1 Data Generation

5.1.1 Dataset Generation

Due to the lack of public datasets for vehicle-specific APIs, we created a proprietary dataset using LLM-driven data augmentation (Gemini Ultra). Entries followed a Q/A format to mimic user commands and API calls. We meticulously validated each entry for consistent formatting and accuracy. Similarly, we created a complex-task dataset chaining two to five API calls via the same process.

5.1.2 API Documentation Generation

To better define the scope of the vehicle system functionalities and distill the model's capabilities, we used Gemini Ultra to generate a comprehensive API documentation in JSON format and verified its validity manually. This documentation outlined all endpoints, functions, and values recognized by the vehicle system. During the second iteration of data generation, relevant sub-categories from this API documentation were incorporated into the prompts. This ensured that newly generated entries remained within the scope of the model's capabilities. Any out-of-scope entries from the initial dataset were removed and replaced.

5.2 LLM Agent for API Generation

5.2.1 Local Model Selection and Quantization

To better simulate the onboard copilot of an electric vehicle, the selected LLMs must be installed locally and capable of running on the T4 GPU provided by Google Colab that offers 14.9 GB of VRAM. Considering the trade-offs between model size and reasoning capabilities, we limit our selection to LLMs with around 7 billion parameters. We choose Mistral 7B, Llama-2 7B, Llama-3 7B, and M7-7B which is a fine-tuned variant of Mistral 7B that ranks higher than original Mistral 7B on the Hugging Face OpenLLM leaderboard. To load these 7B-level LLMs on the T4 GPU, we apply 4-bit quantization to reduce the memory requirements without sacrificing too much performance.

5.2.2 Inference Prompt Engineering

To optimize the LLM's ability to generate valid API calls during inference, we focus on careful prompt engineering. We utilize the API documentation in JSON format and filter it down to relevant subcategories based on the task at hand. This tailored documentation serves as a critical guardrail and reference for the LLM.

We pass the question as the user prompt along with the rest of the prompt to LLMs for API generation. We then use regular expressions to parse the API from the raw output and compare it with the parsed answer from the dataset.

The following shows the input prompt template, and an sample input prompt is in the appendix:

<API Doc filtered by subcategory> + <Instruction> + <Q/A Example 1&2> + <Formatting Rules> + <User Prompt>

5.2.3 Local Model Fine-Tuning

We leveraged fine-tuning to boost the LLMs' performance for our task. However, it was not plausible or effective to fine-tune all parameters with the time and computational resources on hand. We employed a parameter efficiency fine-tuning method using QLoRA. The dimensions of the low-rank

matrices are set to 16 and the alpha value to 16. The maximum sequence length is set to be 2048 and inherited from the pre-trained model. Through these parameters, we can substantially decrease the number of trainable parameters, thus enhancing model efficiency. The temperature of all the models is set to 0 to maintain stability. We also used cross-validation and early stopping to mitigate the problem of overfitting caused by the small dataset.

5.3 Task Decomposition

We performed few-shot evaluation on the best-performing models using a dataset with 50 complex multi-task commands, adding a decomposition module to break down commands into individual tasks. We explored the following methods: pure prompt engineering, naive sentence decomposition, and a LLM submodule. The prompt was constrained to only output the corresponding valid API calls for each task in a command with no further processing on the dataset to examine the model's inherent capabilities of completing complex tasks. Adding a submodule to decompose the input simplifies the problem to a single task and reduces overall complexity. We first evaluated the performance with a naive split on the commands by grammatical rules and punctuation, then compared its performance with that of a LLM as a task decomposer. For consistency, we use the same LLM for both the submodule and evaluation.

5.4 Agentic Reflection Workflow

To enhance the accuracy of API generation, we incorporated an agentic reflection workflow. Following the initial LLM generation of an API call, the model received the prompt, API documentation, and its own prior output along with a new system prompt. This additional information allowed the LLM to self-evaluate the generated API in the context of the task and documentation. By leveraging this self-assessment, the model was then given a second opportunity to refine its output and produce a potentially more accurate API call.

6 Experiments

We designed experiments to evaluate the LLMs' performance on ***Single Task*** - where each given instruction can be fulfilled with a single API call based on the API documentation generated in Section 5.1.2, and also on ***Task Decomposition*** - where each given instruction requires multiple API calls.

6.1 Experimental Setup

6.1.1 Dataset

We utlized the dataset generated as described in Section 5.1. For ***Single task***experiments, we randomly select 50 out of the 120 data entries as the training set for fine-tuning and use the rest as the testing set. Due to resource constraints, we didn't perform QLoRA fine-tuning for LLMs in ***Task Decomposition*** as we don't have a specific training set design for that experiment.

6.1.2 Baseline Methods

To establish a comprehensive performance baseline and quantify the potential performance gap, we benchmarked the capabilities of prominent cloud-based large LLMs, GPT-4 and Gemini Pro, using our evaluation workflow the same as the local small LLMs. We set the temperature parameter of both models to be 0 to maximize output consistency.

6.1.3 Evaluation Protocols

We selected prediction accuracy as the evaluation metric, focusing on the ability of the LLMs to generate correct and executable API calls based on the user prompts. We only consider API calls that 100% match the targets after parsing as true positives in order to simulate the API calling requirements in realistic scenarios.

For ***Single Task***, after splitting the generated dataset as in Section 6.1.1, we tested the performance of the chosen small LLMs and the cloud-based large LLMs with prompts composed from the template provided in Section 5.2.2. We then performed QLoRA fine-tuning to the small LLMs with the training dataset. We additionally performed 10-fold cross-validation for the top 3 small LLMs to evaluate the effectiveness of fine-tuning.

For ***Task Decomposition***, the top-2 performing small LLMs in ***Single Task*** with few-shot prompting were chosen as they have demonstrated better API generation capabilities. We tested the performance of the selected small LLMs as well as the cloud-based LLMs following the same workflow as ***Single Task*** and changing the dataset to complicated instructions.

6.2 Results

Table 1 shows the performance of various models on the ***Single Task*** experiment. Notably:

Llama Limitations: Llama family demonstrates near zero accuracy under few-shot prompting which is caused by its output not following the

LLM	Few-Shot	LoRA (rand)	CV (10-fold)
Mistral 7B	63.3%	67.1%	70.8% $\pm$ 2.4%
Llama-3 8B	0%	72.9%	66.7% $\pm$ 5.6%
M7-7B	53.3%	55.7%	59.2% $\pm$ 2.2%
Llama-2 7B	0%	31.4%	22.5% $\pm$ 4.0%
Llama-2 13B	3.3%	37.1%	/
Gorilla	10.0%	/	/
Gemini-Pro	94.2%	/	/
GPT-4	94.2%	/	/

Table 1: Single Task API Generation Accuracy

given format and being unparseable with regular expressions.

Mistral vs. M7-7B: Mistral 7B has a decent performance under few-shot prompting conditions and is better than M7-7B which has a higher Hugging Face OpenLLM score.This highlights that API generation benefits from domain-specific strengths rather than solely relying on general language capabilities.

Task-specific Gorilla:: Gorilla's poor performance suggests that API-centric training, while beneficial for specific functions, may hinder its language comprehension for broader tasks that require understanding and integrating API documentations and definitions that vary from its training format.

Fine-Tuning Impact: After fine-tuning with QLoRA, all three models have shown improved performance, with Llama-3 8B's accuracy reaching 72.9%, which is the highest among the small LLMs.

Cross-Validation: For Mistral 7B and Llama 3 8B, the top-2 performing LLMs, we also performed a 10-fold cross validation to examine their holistic performance on the entire dataset. We report the results in the last column of Table 1, which supports our observation that Mistral 7B and Llama-3 8B as the top-performing small LLMs.

In ***Task Decomposition*** we found that Gemini-Pro demonstrates a significantly worse performance than GPT-4, which we suspect because Gemini-Pro has much fewer parameters compared to GPT-4. Small LLMs saw performance gains with naive sentence decomposition, revealing limitations in their inherent task-breaking abilities. However, this technique was detrimental to Gemini-Pro, potentially due to context disruption and syntactic breakage. Additionally, many incorrect cases arose from subjective phrasing in queries not exactly matching our dataset, such as "starbucks" versus "starbucks-coffee", likely caused by model hallucinations from complex contexts.

LLM	Few-shot	Sentence Decomp. (naive)	Sentence Decomp. (LLM)
Mistral 7B	45%	55%	46%
Llama-3 8B	22.8%	30%	21.4%
Gemini-Pro	65%	55%	58%
GPT-4	92.2%	/	/

Table 2: Task Decomposition API Generation Accuracy

While fine-tuning improves small LLM performance, they still lag behind GPT-4 and Gemini-Pro. However, considering cost and offline access limitations of cloud-based models, this performance gap may be acceptable for some use cases. More specifically, GPT-4 costs around $5 to generate only 50 API calls. More importantly, cloud-based LLMs become unavailable in the absence of reliable Internet connection.

Additionally, none of the models benefited from the agentic reflection workflow, and instead have shown worse results mainly due to hallucination. Therefore, we judge that agentic reflection is not effective without a model specifically trained to performance objective evaluation.

7 Conclusions and Future Work

In this project, we examined the API generation ability of local small LLMs running on a single consumer–grade GPU. The experimental results demonstrate that the small LLMs, after fine-tuning with limited training data, possess reasonable API generation ability, but are still significantly worse than cloud-based LLMs. However, as our goal is to build a copilot agent for EVs, the local LLMs may serve as a valid backup for the cloud-based large LLMs in certain scenarios where internet connection becomes unavailable.

We also argue that lack of significant performance increases from fine-tuning is attributed to the lack of data: we only have our own synthesized data for this task. A possible direction for future work is further fine-tune these small LLMs with a large dataset and re-evaluate their performances.

8 Division of Labor

8.1 Yuhang Qian

Researched on EV-domain-specific topics to verify the validity of the project's scope. Synthesized and validated dataset and API documentation using LLM-driven data augmentation to address the lack of suitable public datasets. Explored various cloud-based and local LLMs to find suitable candidates for local deployment and benchmarking purposes. Performed prompt engineering for data generation and single task API generation workflows. Performed baseline model performance benchmark with Gemini-Pro. Researched and implemented agentic reflection workflow to further improve model performance.

8.2 Kevin Zheng

Explored small large language models for model selection. Experimented with model setup and fine-tuning with Colab environments. Refined system prompts to improve text generation. Benchmarked Gorilla LLM for single and few-shot API call generation. Evaluated Mistral and LLaMa on agentic reflection workflow. Assisted with writing and editing reports.

8.3 Charlene Yuen

Explored methods for ***Task Decomposition***. Created complex task dataset and system prompts. Experimented with prompt engineering and sentence decomposition methods on complex tasks. Evaluated above methods on Mistral, Llama, and Gemini as a benchmark. Assisted with report writing.

8.4 Tianyu Bai

Initially explored decomposed prompting techniques to perform ***Task Decomposition***. Subsequently, worked on training and fine-tuning LLMs to generating ***Single Task*** API calls. Implemented model fine-tune using supervised fine-tuning technique with ***LoRa*** applied, and model evaluation based on accuracy metrics. To enhance model performance, proposed and applied strategies such as selective categorical data training, early stopping, and cross-validation.

8.5 Enyu Zhao

Explored small large language models for model selection. Experimented with model setup and fine-tuning with Colab environments. Tested the performance of Mistral 7B in ***Task Decomposition.*** Benchmarked the performance of GPT-4 on ***Single Task*** and ***Task Decomposition***. Report writing and organization.

References

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Md. Rafiul Biswas, Ashhadul Islam, Zubair Shah, Wajdi Zaghouani, and Samir Brahim Belhaouari. 2023. Can chatgpt be your personal medical assistant?

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Google. 2024. Gemini: A family of highly capable multimodal models.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 2024. Lora+: Efficient low rank adaptation of large models.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Shimin Li, Tianxiang Sun, Qinyuan Cheng, and Xipeng Qiu. 2024. Agent alignment in evolving social norms.

OpenAI. 2021. Chatgpt: A large-scale generative model for open-domain chat. `https://github.com/openai/gpt-3`.

OpenAI. 2024. Gpt-4 technical report.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models.

Hugo Touvron, Louis Martin, and et al. Kevin Stone. 2023. Llama 2: Open foundation and fine-tuned chat models.

Rayden Tseng, Suzan Verberne, and Peter van der Putten. 2023. *ChatGPT as a Commenter to the News: Can LLMs Generate Human-Like Opinions?*, page 160–174. Springer Nature Switzerland.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2023. A survey on large language model based autonomous agents.

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

Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. 2023. Tidybot: personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102.

Zhiheng Xi, Senjie Jin, Yuhao Zhou, Rui Zheng, Songyang Gao, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Self-polish: Enhance reasoning in large language models via problem refinement.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2023. Memorybank: Enhancing large language models with long-term memory.

9 Appendix

9.1 Detailed System Prompt

9.1.1 API Documentation

We provide a link to it as it won't fit in the ACL template. https://drive.google.com/file/d/1aizZB4vUr4ayQrBxIXsKnxdgRgx_xsPo/view?usp=drive_link.

9.1.2 System Prompt

Single Task:

The link to the .txt file of the prompt: https://drive.google.com/file/d/1JlJj7XN5aF6J3_GF1_suKxU1VG7_wUyO/view?usp=sharing

You are a copilot for an electric vehicle, all you can do is to generate API calls starting with GET. Generate the api call based on the prompt below, don't say anything else, just output the api call starting with GET based on the API documentation above, and these two example below:

Example 1:
Q: Can you play some music for roadtrips?
A:
GET /vehicle/infotainment/music?mood=roadtrip

Example 2:
Q: Navigate to Yellow Stone National Park.
A:
GET /vehicle/navigation/destination? address=yellow+stone+national+park

Some formatting rules include:

1. if the prompt includes spaces, replace it with "+", so "Los Angeles" becomes "los+angeles"
2. Use lower case only for parameter values
3. Don't include anything else, just generate the api call starting with GET

This is the API format and the category details:
GET /vehicle/<category>/<function>

Task Decomposition:

The link to the .txt file of the prompt: https://drive.google.com/file/d/1JDOQjPZrhW5mUYCvNeXEuq6QTzliwzPy/view?usp=sharing

You are a copilot for an electric vehicle, all you can do is to generate API calls starting with GET. Generate the api call based on the prompt below, don't say anything else, just output the api calls starting with GET based on the API documentation above.

You will need to provide a list of API calls separated by commas as a response to a series of questions and commands.

Ensure that you only include valid API calls and parameters from the documentation.

Here are two examples below:

Example 1:
Q: "Play some jazz music and increase the volume."
A:
GET /vehicle/infotainment/music?genre=jazz,
GET /vehicle/infotainment/volume?control=increase

Example 2:
Q: Activate the fog lights, search for the nearest charging station, and get the tire pressure.
A: GET /vehicle/control/exterior?action=foglight-on,
GET /vehicle/navigation/search?query=charging+station&sort=current,
GET /vehicle/status/tires

Some formatting rules include:

1. if the prompt includes spaces, replace it with "+", so "Los Angeles" becomes "los+angeles"
2. Use lower case only for parameter values
3. Don't include anything else, just generate the api call starting with GET
4. Use the exact wording as in the question for the values for parameters when applicable.
5. When multiple commands result in multiple parameters for the same endpoint and function, include them in the same API call.
6. Do not include any irrelevant API calls not in the question.

This is the API format and the category details:
GET /vehicle/<category>/<function>,GET /vehicle/<category>/<function>,....,GET /vehicle/<category>/<function>