LaMPilot: An Open Benchmark Dataset for Autonomous Driving with Language Model Programs

Yunsheng Ma[1*], Can Cui[1*], Xu Cao[2*], Wenqian Ye[3], Peiran Liu[1], Juanwu Lu[1],
Amr Abdelraouf[4], Rohit Gupta[4], Kyungtae Han[4], Aniket Bera[1], James M. Rehg[2], Ziran Wang[1]
[1]Purdue University [2]University of Illinois Urbana-Champaign
[3]University of Virginia [4]InfoTech Labs, Toyota Motor North America
{yunsheng, ziran}@purdue.edu

Autonomous driving (AD) has made significant strides in recent years. However, existing frameworks struggle to interpret and execute spontaneous user instructions, such as "overtake the car ahead." Large Language Models (LLMs) have demonstrated impressive reasoning capabilities showing potential to bridge this gap. In this paper, we present LaMPilot, a novel framework that integrates LLMs into AD systems, enabling them to follow user instructions by generating code that leverages established functional primitives. We also introduce LaMPilot-Bench, the first benchmark dataset specifically designed to quantitatively evaluate the efficacy of language model programs in AD. Adopting the LaMPilot framework, we conduct extensive experiments to assess the performance of off-the-shelf LLMs on LaMPilot-Bench. Our results demonstrate the potential of LLMs in handling diverse driving scenarios and following user instructions in driving. To facilitate further research in this area, we release our code and data at GitHub.com/PurdueDigitalTwin/LaMPilot.

1. Introduction

Autonomous driving (AD) has witnessed remarkable progress in recent years, with an increasing number of commercial autonomous vehicles (AVs) being deployed on public roads [30]. State-of-the-art AD systems can be broadly classified into two categories: 1) a modular approach, where standalone models are developed for perception, prediction, and planning independently [15], and 2) an end-to-end approach that directly maps sensor data to control signals via a single neural network [16, 18, 38]. Despite significant breakthroughs, both approaches struggle to handle arbitrary user commands effectively, such as "overtake the car in front of me."

Large Language Models (LLMs) have demonstrated impressive capabilities in language comprehension and reasoning [42, 61], showing potential to improve the safety, explainability, and user-friendliness of AVs. Utilizing LLMs to solve AV-related tasks is gaining momentum [9, 13, 24, 60, 65]. However, the integration of LLMs into existing AD frameworks presents several challenges. Firstly, there is a lack of well-established paradigms for incorporating LLMs into the decision-making process of AVs. Secondly, there is a shortage of benchmarks designed to evaluate and compare the performance of LLM-based agents in the context of driving.

To address these challenges, we propose a novel framework called LaMPilot. Inspired by Code as Policy [25], which utilizes code LLMs to write robot policy code, LaMPilot employs Language Model Programs (LMPs) as the action space instead of low-level vehicle control signals. We equip LLMs with APIs that cover various functional primitives, enabling them to connect natural language instructions to executable driving plans through code generation.

We also introduce LaMPilot-Bench, the first benchmark for evaluating LMPs in AD. The primary objective for agents operating within LaMPilot-Bench is to accomplish assigned tasks safely and efficiently. LaMPilot-Bench incorporates an interactive simulator and evaluator, featuring programmatic scoring mechanisms to assess policy performance. The main contributions of this work are summarized as follows:

- We propose LaMPilot, a novel framework that integrates LLMs into autonomous driving systems, enhancing their ability to interpret and follow user commands.
- We introduce LaMPilot-Bench, the first benchmark dataset designed to evaluate the performance of LLM-powered agents in autonomous driving. Each scenario in LaMPilot-Bench consists of a task described in natural language, along with a simulated environment for comprehensive evaluation.
- Adopting the LaMPilot framework, we conduct extensive experiments to assess the performance of off-the-shelf

*Equal Contribution

LLMs on LaMPilot-Bench. Our results demonstrate the great potential of LLMs in handling diverse driving scenarios and following user instructions.

2. Related Works

2.1. Language-Guided Driving

Recent studies highlight the potential role of language in enhancing self-driving technology. Companies like Wayve are integrating natural language to improve the learning and explainability of their driving models. For example, their LINGO-1 system combines vision, language, and action modalities [29, 53].

Researchers have also been incorporating LLMs into AD systems for various purposes, such as generalization [4, 8, 11, 12, 19, 23, 27, 28, 35, 44, 49, 55, 56, 66], interpretability [10, 31, 36, 57, 63], and human-vehicle interaction [6, 7, 39, 40, 49, 59]. DiLu [56], for instance, instills knowledge-driven capability into AD systems by considering how humans drive. It introduces a new framework that includes an interactive environment, a driver agent, and a memory component. LMDrive [39] proposes an end-to-end, language-based AD framework that processes camera-LiDAR sensor data, comprehends natural language driving instructions, and directly generates vehicle control signals.

However, most existing works either provide high-level goals, let LLMs generate the ego vehicle's future trajectories, or involve detailed vehicle control signals like throttle and steering angles. In contrast, our work uniquely uses LLMs to generate program code as driving policies, which are then executed with classical planners or controllers. Our approach takes advantage of the strengths of both LLMs and traditional AD components.

2.2. Large Language Models for Code Generation

The emergence of powerful LLMs has opened up new possibilities for leveraging programming languages to perform various tasks across multiple domains [58]. Code generation [64] is one of the most prominent applications, where LLMs are used to generate code snippets based on natural language descriptions. LLMs have also been employed for information extraction tasks using code as an intermediate representation [48, 50]. The robotics domain has also benefited from the integration of LLMs and code. Code as Policy [25], ProgPrompt[41], and VoxPoser [17] demonstrate how LLMs can generate code to control robots and manipulate objects in 3D environments. In the vision domain, works such as ViperGPT [43] showcase the potential of using LLMs to generate code for visual processing and reasoning. Moreover, code provides a foundation for logical planning [26, 51], which can be leveraged by LLMs to tackle complex, long-horizon tasks. The combination of LLMs and code has revealed a new paradigm with broad applicability across numerous engineering and scientific domains.

3. LaMPilot Methodology

This section elaborates on the framework of LaMPilot, which integrates the world knowledge and reasoning capabilities of LLMs into an AD system to enable user instruction following. LaMPilot uses LLMs to generate LMPs that serve as the action space. LMPs have the inherent potential to symbolize actions that are both temporally generalizable and hierarchically structured. For instance, a complex maneuver like overtaking can be programmatically split into a series of actions, including lane changes, acceleration, and possibly another lane change, composed with conditional logic.

LaMPilot is designed as an add-on module for existing AD systems rather than as a standalone solution. It synergizes the emerging capabilities of LLMs with the proven efficiency of classical AD algorithms. Instead of directly issuing real-time, low-level control signals, the LLM generates code snippets at lower frequencies. These snippets, formulated as temporary policies, guide the strategic navigation of the ego vehicle based on user instructions.

3.1. Prompts

As shown in Fig. 1, the LaMPilot framework takes inputs including API documentation ($\mathbf{A}$), human instructions ($\mathbf{I}$), and driving context ($\mathbf{C}$). It leverages Chain-of-Thought (CoT) reasoning [54] to write policy code ($\mathbf{P}$). This process can be formulated as:

$$[\mathbf{R}, \mathbf{P}] = \ell([\mathbf{A}, \mathbf{I}, \mathbf{C}]) \tag{1}$$

where ℓ represents the LLM backbone.

An example to demonstrate the code generation process is provided in Tab. 1. The instruction is a spontaneous command from the user. In addition, we also provide environment contexts and API documentation as part of the input prompt for the LLM.

The driving contexts include relevant information about the driving environment. We develop an interface that converts the numerical vector data in simulation into a narrative format using a structured language generator [4]. This narrative provides a natural input interface for the LLM and does not require additional fine-tuning for interpretation.

3.2. Functional Primitives

Policy code involves calls to functional primitives. They are implemented with classical planning and control algorithms. Due to the autoregressive nature of LLMs, generating long completions can introduce significant latency, making them less suitable for time-critical tasks like collision avoidance. The main idea is to utilize LLMs for

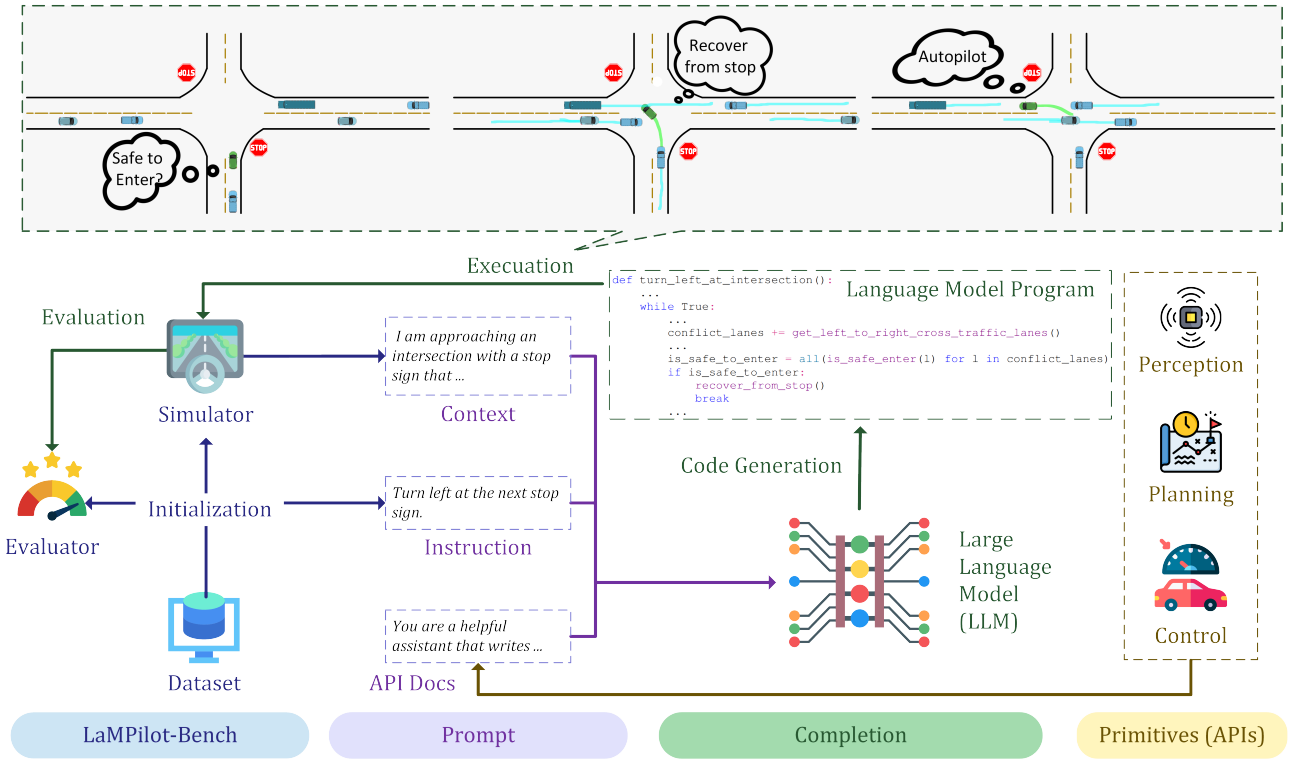

Figure 1. An overview of the LaMPilot framework. The Large Language Model (LLM) receives a prompt containing human instructions, driving context, and API documentation. It then writes language model programs that serve as driving policies. These policies are executed in the simulator to complete the specified driving task and are subsequently evaluated by the evaluator to assess the effectiveness of the generated policy code.

strategic planning while intentionally avoiding their direct involvement in low-level control tasks.

Inspired by the concept of Responsibility-Sensitive Safety [37], which ensures that the autonomous driving system will not issue a command that would lead to an accident, we include safety heuristics in our API implementations to minimize risks. Our API suite is categorized into four main types:

- Ego APIs: They provide information about the status of the ego vehicle.
- Perception APIs: They include functions like object and lane detection, which can be called to acquire information about the surrounding environment.
- Navigation APIs: They provide route planning functionalities given a target destination.
- Control APIs: They translate LLM-generated code into low-level control signals for the vehicle.

In the event of an exception during policy execution, the ego vehicle automatically switches to a predefined `autopilot` mode to prevent undefined behaviors. To provide LLMs with essential information about the available APIs and guide their correct usage, we include the API Documentation ($\mathbf{A}$) in the prompt. These documents include not only the input and output specifications but also provide descriptions of their usage and the underlying logic.

3.3. Completions

The completion produced by the LLMs is anticipated to be valid functions, written with the provided APIs. These functions could range from straightforward, one-off functions to more complex generator functions. A simple example is demonstrated below, where the target speed of the ego vehicle is altered:

```
def decrease_speed_by_5():
    set_target_speed(get_target_speed() - 5)
```

Moreover, LaMPilot supports the use of control structures such as *if-else* and *loop* statements, enabling the LLM to create dynamic feedback policies. An example of this is shown in Tab. 1. In this sample, the LLM generates a policy function `overtake_using_right_lane`. The function first checks if there is a right lane available and a vehicle in front of the ego vehicle. If both conditions are met, it enters a loop to wait for a safe opportunity to change into the right lane. Once in the right lane, the function enters another loop to monitor the distance to the target vehicle until it has successfully overtaken the target vehicle.

API Docs (A)
You are a helpful assistant that writes python code to complete any autonomous driving task specified by me.
Here are the APIs you can use: ... **[omitted]**
Instruction (I)
Go around the car in front of you using the right lane.
Driving Context (C)
My current speed is 31.1 m/s. I am driving on a highway with 2 lanes in my direction, and I am in the 2nd lane from the right. There is a car in front of me in my lane, at a distance of 44.9 m, with a speed of 31.1 m/s.

Chain-of-Thought Reasoning (R)
1) Check if there is a right lane available. 2) Check if there is a vehicle in front of the ego vehicle in the current lane within 100 meters. 3) If so, check if it is safe to enter the right lane. 4) If it is safe to enter the right lane, set the target lane of the ego vehicle to the right lane. 5) Monitor the distance to the target vehicle. 6) If the target vehicle is still in front of the ego vehicle, set the target speed of the ego vehicle to 1.5 times the speed of the target vehicle to overtake. 7) Repeat steps 5-6 until the target vehicle is no longer in front of the ego vehicle.
Code as Policies (P)

```
def overtake_using_right_lane():
    ego_vehicle = get_ego_vehicle()
    current_lane = get_lane_of(ego_vehicle)
    right_lane = get_right_lane(ego_vehicle)
    target_vehicle = detect_front_vehicle_in(current_lane)
    if right_lane is None:
        say("There is no right lane to change into.")
        return
    if target_vehicle is None:
        say("There is no vehicle in front of me.")
        return
    # Check if it is safe to enter the right lane
    while True:
        if is_safe_enter(right_lane):
            set_target_lane(right_lane)
            break
        yield autopilot()
    # Monitor the distance to the target vehicle
    while True:
        distance_to_target = get_distance_between_vehicles(ego_vehicle, target_vehicle)
        if distance_to_target < 0:  # The target vehicle is still in front of the ego vehicle
            set_target_speed(get_speed_of(target_vehicle) * 1.5)
            yield autopilot()
        else:
            break
```

Table 1. This example shows the chain-of-thought reasoning and code generation process of LaMPilot. Contents with `blue` titles are generated by LLMs.

The `yield` statements allow the AD system to take control and execute the low-level actions required to follow the high-level policy set by the LLM.

This example demonstrates how LaMPilot leverages the reasoning capabilities of LLMs to generate dynamic, context-aware driving policies. By combining the strategic decision-making of LLMs with the real-time execution of classical AD algorithms, LaMPilot enables more flexible instruction following for AVs.

4. LaMPilot Benchmark

We introduce LaMPilot-Bench, the first benchmark for evaluating the instruction-following capabilities of LLM-based agents in AD. As shown in Fig. 1, LaMPilot-Bench consists of three key components: a simulator, a dataset, and an evaluator.

4.1. Simulator

The LaMPilot-Bench simulator is built upon HighwayEnv [21], a widely used platform for AD research and tactical decision-making. HighwayEnv offers various driving models and simulates realistic multi-vehicle interactions. We extend HighwayEnv with interfaces suitable for LLM-based agents and implement custom intersections to diversify the driving scenarios.

4.2. Dataset

The LaMPilot dataset consists of 4,900 semi-human-annotated traffic scenes, with a subset of 500 samples split as the test set. Each data sample includes:

- An instruction **I**: a high-level task description.
- An initial state: used to initialize the simulator.
- Goal state criteria: aligned with the instruction **I**.

The dataset covers diverse driving scenarios, as shown

Statistic	Value
Total scenarios	4,900 (100%)
Distance-related	1,200 (24.5%)
Speed-related	1,200 (24.5%)
Pulling over	200 (4.1%)
Routing	1,500 (30.2%)
Lane changing	400 (8.2%)
Overtaking	400 (8.2%)
Highway	3,400 (69.4%)
Intersection	1,500 (30.6%)
Average instruction length	7.6 words
Maximum instruction length	14 words
Minimum instruction length	2 words

Table 2. Main statistics of the LaMPilot-Bench dataset.

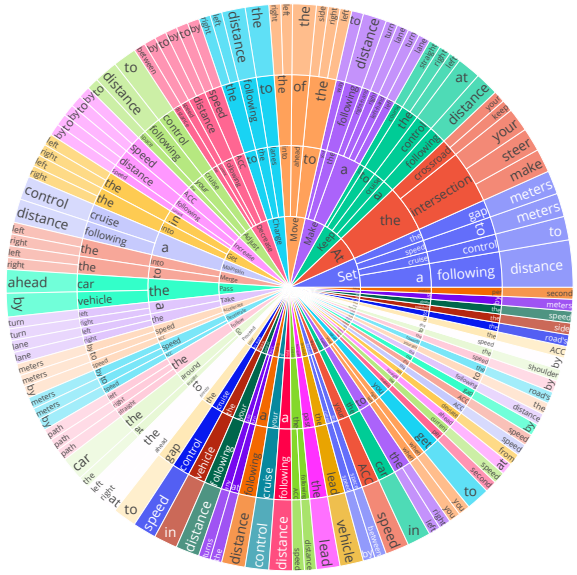

Figure 2. Distribution of instructions in the LaMPilot-Bench dataset.

in Table 2. For each driving scenario, LaMPilot includes various situations. Taking turning scenarios as an example, the diversity is reflected in several variables, such as the ego vehicle's initial position and state, the specific task (turning left/right or going straight), the number of other vehicles, and their positions and states. The driving model parameters for other vehicles are randomly initialized, and each scenario is assigned a random seed.

The dataset also contains a variety of instructions reflecting realistic in-cabin human instructions categorized by maneuver types (e.g., routing, lane changing, overtaking) and scenario types (highway and intersection). Fig. 2 shows the distribution of the first four words in the instructions, highlighting their diversity. Main statistics are provided in Tab. 2.

4.3. Evaluator

The LaMPilot-Bench evaluator incorporates metrics to assess the safety and efficiency of the agent driving policies.

Safety Metric Time-to-collision (TTC) is used to measure the vehicle's ability to maintain safe distances and avoid collisions. In a scenario with $n+1$ vehicles (ego vehicle labeled as 0), the TTC (τ_i) with vehicle i is calculated using the ego vehicle's velocity $\mathbf{v}_0$ and position $\mathbf{p}_0$, and vehicle i's velocity $\mathbf{v}_i$ and position $\mathbf{p}_i$ ($1 \leq i \leq n$):

$$\tau_i = -\frac{(\mathbf{p}_0 - \mathbf{p}_i) \cdot (\mathbf{v}_0 - \mathbf{v}_i)}{\|\mathbf{v}_0 - \mathbf{v}_i\|^2} \tag{2}$$

The minimum positive TTC value $\tau_{\min}$ among all n vehicles over all time steps t (up to task completion time T) represents the nearest potential collision:

$$\tau_{\min} = \min_{1 \leq i \leq n,\, 1 \leq t \leq T} \tau_i^t \tag{3}$$

TTC scores are empirically assigned based on a 2-second safety margin, with values above 2 seconds considered safe and given a score of 100:

$$\text{TTC} = \begin{cases} 100 & \text{if } \tau_{\min} > 2 \\ \max(0, 100 - \frac{1}{\tau_{\min}}) & \text{if } 0 < \tau_{\min} \leq 2 \\ 0 & \text{if } \tau_{\min} \leq 0 \end{cases} \tag{4}$$

Speed variance (SV) is considered as another safety metric [14]. The ego vehicle's speed standard deviation σ_0 is calculated as:

$$\sigma_0 = \sqrt{\frac{\sum_{t=1}^{T} (\|\mathbf{v}_0^t\| - \mu_0)^2}{T}}, \tag{5}$$

where μ_0 is the mean speed:

$$\mu_0 = \frac{\sum_{t=1}^{T} \|\mathbf{v}_0^t\|}{T}. \tag{6}$$

The SV score is defined as:

$$\text{SV} = \max(0, 100 \cdot (1 - \sigma_0 / \sigma_{\text{safe}})), \tag{7}$$

where σ_{safe} is the maximum safe speed deviation, determined empirically.

Efficiency Metric The time efficiency (TE) score evaluates the policy's ability to complete the task within a predefined time limit T_{limit}:

$$\text{TE} = \max(0, 100 \cdot (1 - T / T_{\text{limit}})) \tag{8}$$

A TE score closer to 100 indicates a more efficient policy.

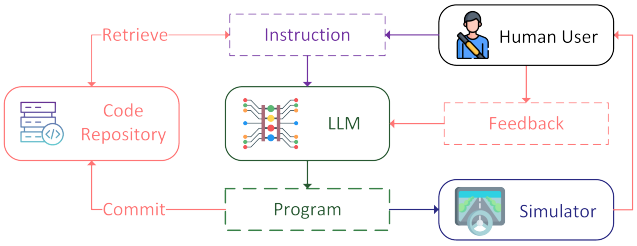

Figure 3. Schematic view of the human feedback baseline based on retrieval-augmented generation.

Task Completion Criteria A task is considered successfully completed when the agent achieves the objectives specified in the instruction while maintaining safety (i.e., avoiding collisions) and efficiency (i.e., finishing within the allotted time). For example, a lane change task is completed when the vehicle is in the target lane with its heading aligned with the lane's direction within a specified threshold.

Overall Scoring The final score aggregates all individual metrics, weighted according to their importance:

$$\text{Score} = W_{\text{TTC}} \cdot \text{TTC} + W_{\text{SV}} \cdot \text{SV} + W_{\text{TE}} \cdot \text{TE} \quad (9)$$

5. Experiments and Results

Our experiments are designed to establish baselines for the proposed LaMPilot-Bench. The main objectives are to assess the performance of LLM-based agents in interpreting human instructions within driving contexts and to evaluate the capability of LLMs to generate code for motion planning using provided functional primitives.

5.1. Experiment Setup

Models We conduct benchmarking using various state-of-the-art large language models, which include both open-source and proprietary solutions[1]:

- ***Llama 2*** [45] (`llama-2-70b-chat`)
- ***PaLM 2*** [1] (`code-bison`)
- ***ChatGPT*** [34] (`gpt-3.5-turbo`)
- ***GPT-4*** [33] (`gpt-4`)
- ***GPT-4-Turbo*** [32] (`gpt-4-1106-preview`)

Code Execution To execute a LMP, the Python `exec` function is used. It takes the LMP code as an input string, along with two dictionaries that define the execution scope: (i) `apis`, which includes all APIs the code may invoke, and (ii) `policies`, an initially empty dictionary that will eventually contain a `policy` variable. If the LMP is designed to return a generator, this generator is extracted from the `policies` dictionary after the execution of the `exec` function.

[1]OpenAI API accessed in October and November 2023.

Evaluation We set a 60-second time limit for each scenario. If a task is not completed within this time frame, the test case is considered a failure, and the simulation is terminated, resulting in a driving score of 0. For successful cases, the driving score is calculated as follows:

$$\text{Driving Score} = \frac{\alpha}{N_s} \sum_{i=1}^{N_s} \text{Score}_i - \beta \cdot P_{\text{collision}} \quad (10)$$

Here, N_s represents the number of successful test cases, α is the success rate (ranging from 0 to 1), β is the collision rate (also between 0 and 1), and $P_{\text{collision}}$ is the penalty factor for collisions (set at 500 in our experiments). The Score_i for each individual sample is calculated based on Eq. (9), with weights W_{TTC}, W_{SV}, and W_{TE} set to 0.5, 0.3, and 0.2, respectively. The driving score provides a comprehensive assessment of the agent's performance in LaMPilot-Bench.

5.2. Baselines

5.2.1 Heuristic Baselines

In autonomous driving, rule-based models are favored for their deterministic and interpretable nature. In our experiments, we employ two rule-based baseline policies: the Intelligent Driver Model (IDM)[46] and the Minimizing Overall Braking Induced by Lane Changes (MOBIL) principle[20]. IDM describes a rule for updating the acceleration of a vehicle to avoid collisions based on the proximity and relative velocity of the vehicle to the object directly in front. MOBIL is an extension of IDM where a lane change is executed if the prospective new lane offers a more favorable driving scenario and the maneuver can be conducted safely. These baselines can be considered as operating on *random chance*, as the policy is independent of user instructions but follows predefined rules. Additionally, we include a *human performance* baseline, where a licensed human driver controls the vehicle in the simulation using arrow keys on the keyboard, following the commands and visuals displayed. This baseline provides a reference for human-level performance on LaMPilot-Bench.

5.2.2 Zero-Shot and Few-Shot Baselines

We also include the zero-shot and few-shot chain-of-thought (CoT) prompting [54] as baseline methods. The zero-shot CoT baseline leverages the prompt *"How to complete the task step by step."* In the few-shot setting, we follow the standard practice [2] by including 3 human-written code exemplars before the test instance. These in-context

Method	Learning	Collision (↓)	Completion (↑)	TTC Score (↑)	SV Score (↑)	TE Score (↑)	Driving Score (↑)
Heuristic Baselines							
IDM	N/A	0.0%	20.4%	92.8	80.9	71.0	17.3
MOBIL		0.0%	15.3%	82.8	85.9	46.7	11.7
Human Driver		0.0%	98.0%	93.4	86.3	81.3	84.6
LLM-based Agents							
Llama 2	0-Shot	0.0%	20.4%	92.8	81.4	68.9	17.3
PaLM 2		3.1%	35.7%	78.6	76.5	78.8	12.8
ChatGPT		1.0%	40.8%	83.9	75.5	74.0	27.8
GPT-4		4.1%	72.4%	61.3	70.5	74.0	28.5
GPT-4 Turbo		3.1%	74.5%	73.2	70.1	73.9	39.1
Llama 2	3-Shot	1.0% (+1.0)	63.3% (+42.9)	68.3 (-24.5)	71.4 (-10.0)	73.8 (+4.9)	39.6 (+22.3)
PaLM 2		3.1% (0.0)	71.4% (+35.7)	73.4 (-5.2)	69.7 (-6.8)	72.0 (-6.8)	36.6 (+23.8)
ChatGPT		4.1% (+3.1)	83.7% (+42.9)	70.9 (-13.0)	73.7 (-1.8)	77.7 (+3.7)	41.2 (+13.4)
GPT-4		1.0% (-3.1)	84.7% (+12.3)	69.4 (+8.1)	72.0 (+1.5)	76.7 (+2.7)	55.9 (+27.4)
GPT-4 Turbo		1.0% (-2.1)	80.6% (+6.1)	69.1 (-4.1)	71.5 (+1.4)	74.9 (+1.0)	52.6 (+13.5)
Llama 2	Human Feedback	0.9% (-0.1)	32.7% (-30.6)	83.9 (+15.6)	74.6 (+3.2)	74.6 (+0.8)	21.4 (-18.2)
PaLM 2		0.0% (-3.1)	45.5% (-25.9)	81.1 (+7.7)	74.4 (+4.7)	77.6 (+5.6)	35.6 (-1.0)
ChatGPT		0.9% (-3.2)	80.9% (-2.8)	70.2 (-0.7)	71.7 (-2.0)	77.5 (-0.2)	54.2 (+13.0)
GPT-4		0.9% (-0.1)	92.7% (+8.0)	67.6 (-1.8)	72.2 (+0.2)	76.8 (+0.1)	64.0 (+8.1)
GPT-4 Turbo		0.9% (-0.1)	87.3% (+6.7)	74.1 (+5.0)	73.4 (+1.9)	75.5 (+0.6)	60.5 (+7.9)

Table 3. Comparison of baselines on LaMPilot-Bench under different learning settings. Arrows (↑ / ↓) indicate higher or lower scores are better, respectively. Numbers in parentheses show the performance change compared to the previous setting, with (green/red) indicating performance improvements and degradation, respectively.

examples help the model adapt to the tasks in LaMPilot-Bench. The in-context examples are created by a programmer proficient in Python. They are provided with API descriptions and are allowed to write and test their code using the non-test scenarios. The same set of three examples are used for all test cases. The API Docs are also the same across all test cases for both zero-shot and few-shot settings.

5.2.3 Human Feedback Baselines

LLMs have demonstrated proficiency in generating coherent solutions across various tasks without requiring additional fine-tuning. However, when it comes to code generation, especially for complex scenarios, they can produce suboptimal results. The autoregressive nature of these models poses a significant challenge, as tokens generated early in a sequence cannot be modified in the same iteration. This constraint limits the models' ability to refine initial responses, potentially impacting the effectiveness of the generated code [47].

To address these challenges and enhance the performance of LLMs on tasks in LaMPilot-Bench, we introduce a human-in-the-loop approach, where the LLMs serve not only as planners but also as engines for incorporating human feedback. As shown in Fig. 3, human feedback is infused into the learning process, transforming the LaMPilot framework from an open-loop system into an evolving feedback loop. After an LLM generates a policy program ($\mathbf{P}$), it can integrate human feedback ($\mathbf{F}$) with context-specific guidance, enabling the LLM to refine its output.

This approach does not involve fine-tuning or training a new LLM. Instead, we leverage the RAG approach [22] and introduce a code repository module, which is implemented with the Chroma [5] vector database. This repository allows for the storage and retrieval of effective code snippets to be used in similar situations. Following Voyager [47], we use LLM-generated function descriptions, which are then converted into vectors to serve as database keys. These keys are paired with the corresponding function codes as their values. If the human is satisfied with the policy, the code is committed to the repository for future reference. The human feedback learning process is conducted on the non-test set, involving interaction with 100 random samples for each method to ensure fair comparisons.

5.3. Quantitative Results

In this section, we present the experimental results on the LaMPilot-Bench, summarizing the performance of various methods, including heuristic baselines, 0-shot and 3-shot baselines, and human feedback baselines. These results are shown in Tab. 3.

Rule-Based Methods We first evaluate the performance of rule-based approaches, specifically the IDM and MOBIL algorithms. Both methods achieve a zero collision rate in LaMPilot-Bench. However, it is important to note that these methods operate independently of the provided instructions. Without considering human instructions, IDM and MOBIL achieve success rates of 20.4% and 15.3%, respectively. These results provide a reference point for as-

sessing the effectiveness of LLM-based agents in following human instructions.

LLM-Based Agents We assess the effectiveness of off-the-shelf LLMs to reason in the context of autonomous driving and following human instructions. In the 0-shot setting, GPT models and PaLM 2, given only the API Docs, driving context, and instructions, achieve an obvious performance advantage compared to rule-based methods in terms of task completion. This improvement can be attributed to the LLMs' ability to understand and generate code based on natural language instructions, leveraging their extensive pretraining on diverse data sources. However, the 0-shot setting also results in an increase in the collision rate (1%-4%), indicating that the generated code policies do not fully capture the complexities of driving tasks.

When provided with training examples that include exemplar code (3-shot setting), all the evaluated LLMs exhibit notable improvements in completion rates. For instance, Llama 2's completion rate increases from 20.4% in the 0-shot setting to 63.3% in the 3-shot setting. This improvement demonstrates the effectiveness of few-shot learning in adapting LLMs to specific task domains, as the exemplar code provides a clearer understanding of the desired behavior and API usage patterns.

The integration of GPT models with human-in-the-loop learning further enhances their performance in driving tasks. Notably, GPT-4 achieves the highest driving score of 64.0. This improvement highlights the great potential of LLMs in following instructions in the driving context when combined with human feedback. The human feedback loop allows for iterative refinement of the generated policies, enabling the LLMs to learn from their mistakes and continuously incorporate human knowledge to generate more effective driving strategies.

5.4. Discussion

Safety Safety is a critical factor for any AD system [52]. In our context, ensuring that the system does not execute an unsafe action, even if instructed to do so by the human, is essential for responsible autonomy. Our framework includes safety checks at two levels: (1) When the LLM generates code based on human instructions, it first reasons about the safety of the action in the context of the current driving environment. By leveraging the vast knowledge and reasoning capabilities of LLMs, the system can identify potentially unsafe instructions and generate alternative actions that prioritize safety. (2) Even if the LLM fails to catch an unsafe instruction, we have a second layer of safety checks built into the APIs that interface with the vehicle actuators. These APIs encapsulate classical planning and control algorithms with hard safety constraints. This level of safety check serves as a fail-safe mechanism, ensuring that the system does not execute actions that violate safety constraints, regardless of the LLM's output.

Instruction Tuning Instruction tuning has been validated as a technique for LLMs to acquire domain-specific knowledge efficiently [3]. Fine-tuning LLMs on driving-related tasks could enhance their performance and reduce collision rates by providing them with more targeted knowledge specific to driving. However, this may also jeopardize their generalization capability when there exist some spurious correlations in the finetuning datasets (e.g. between driving context and policy) [62]. Therefore, we hope that LaMPilot-Bench can provide a platform to encourage future researchers to explore generalizable solutions to enhance LLMs' abilities in driving tasks.

Multimodal LLMs Exploring the integration of vision modality in multimodal LLMs for AD tasks is another important future direction. Our current framework relies on textual input and output, which limits its ability to perceive and understand the visual world. By incorporating visual perception and understanding capabilities, multimodal LLMs could process and reason about real-world driving scenes more effectively.

6. Conclusion

In this paper, we introduced LaMPilot, a novel framework that integrates Large Language Models (LLMs) into autonomous driving (AD) systems, enabling human-like interaction in diverse driving contexts. We also created LaMPilot-Bench, a benchmark for evaluating the instruction-following capabilities of LLM-based agents in AD. Our experiments demonstrated that off-the-shelf LLMs can generate code policies for driving tasks based on human instructions. However, the notable collision rate indicates the need for further research to fully capture the complexities and safety requirements of real-world driving scenarios. LaMPilot provides a starting point for leveraging LLMs in solving AD-related tasks. As LLMs continue to evolve and integrate with more modalities, we envision a future where they can significantly contribute to the development of safe, efficient, and user-friendly autonomous vehicles.

Acknowledgment

This work is partially funded by the Digital Twin Roadmap of InfoTech Labs, Toyota Motor North America. The contents of this paper only reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views of Toyota Motor North America.

References

[1] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. PaLM 2 Technical Report. *arXiv*, 2023. 6

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, 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. In *NeurIPS*, 2020. 6

[3] Xu Cao, Tong Zhou, Yunsheng Ma, Wenqian Ye, Can Cui, Kun Tang, Zhipeng Cao, Kaizhao Liang, Ziran Wang, James M. Rehg, and Chao Zheng. MAPLM: A Real-World Large-Scale Vision-Language Dataset for Map and Traffic Scene Understanding. In *CVPR*, 2024. 8

[4] Long Chen, Oleg Sinavski, Jan Hünermann, Alice Karnsund, Andrew James Willmott, Danny Birch, Daniel Maund, and Jamie Shotton. Driving with LLMs: Fusing Object-Level Vector Modality for Explainable Autonomous Driving. In *ICRA*, 2024. 2

[5] Chroma. The AI-native open-source embedding database, 2023. URL https://github.com/chroma-core/chroma. 7

[6] Can Cui, Zichong Yang, Yupeng Zhou, Yunsheng Ma, Juanwu Lu, Lingxi Li, Yaobin Chen, Jitesh Panchal, and Ziran Wang. Large Language Models for Autonomous Driving: Real-World Experiments. *arXiv*, 2023. 2

[7] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, and Ziran Wang. Drive as You Speak: Enabling Human-Like Interaction with Large Language Models in Autonomous Vehicles. In *WACVW*, 2024. 2

[8] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, and Ziran Wang. Receive, Reason, and React: Drive as You Say with Large Language Models in Autonomous Vehicles. *IEEE Intelligent Transportation Systems Magazine*, 2024. 2

[9] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, Yang Zhou, Kaizhao Liang, Jintai Chen, Juanwu Lu, Zichong Yang, Kuei-Da Liao, Tianren Gao, Erlong Li, Kun Tang, Zhipeng Cao, Tong Zhou, Ao Liu, Xinrui Yan, Shuqi Mei, Jianguo Cao, Ziran Wang, and Chao Zheng. A Survey on Multimodal Large Language Models for Autonomous Driving. In *WACVW*, 2024. 1

[10] Xinpeng Ding, Jianhua Han, Hang Xu, Wei Zhang, and Xiaomeng Li. HiLM-D: Towards High-Resolution Understanding in Multimodal Large Language Models for Autonomous Driving. *arXiv*, 2023. 2

[11] Daocheng Fu, Wenjie Lei, Licheng Wen, Pinlong Cai, Song Mao, Min Dou, Botian Shi, and Yu Qiao. LimSim++: A Closed-Loop Platform for Deploying Multimodal LLMs in Autonomous Driving. *arXiv*, 2024. 2

[12] Daocheng Fu, Xin Li, Licheng Wen, Min Dou, Pinlong Cai, Botian Shi, and Yu Qiao. Drive Like a Human: Rethinking Autonomous Driving with Large Language Models. In *WACVW*, 2024. 2

[13] Haoxiang Gao, Yaqian Li, Kaiwen Long, Ming Yang, and Yiqing Shen. A Survey for Foundation Models in Autonomous Driving. *arXiv*, 2024. 1

[14] Nicholas J. Garber and Ravi Gadiraju. Factors affecting speed variance and its influence on accidents. *Transportation research record*, 1989. 5

[15] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 2020. 1

[16] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. Planning-oriented Autonomous Driving. In *CVPR*, 2023. 1

[17] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. In *CoRL*, 2023. 2

[18] Bo Jiang, Shaoyu Chen, Qing Xu, Bencheng Liao, Jiajie Chen, Helong Zhou, Qian Zhang, Wenyu Liu, Chang Huang, and Xinggang Wang. VAD: Vectorized Scene Representation for Efficient Autonomous Driving. In *ICCV*, 2023. 1

[19] Ye Jin, Xiaoxi Shen, Huiling Peng, Xiaoan Liu, Jingli Qin, Jiayang Li, Jintao Xie, Peizhong Gao, Guyue Zhou, and

Jiangtao Gong. SurrealDriver: Designing Generative Driver Agent Simulation Framework in Urban Contexts based on Large Language Model. *arXiv*, 2023. 2

[20] Arne Kesting, Martin Treiber, and Dirk Helbing. General Lane-Changing Model MOBIL for Car-Following Models. *Transportation Research Record*, 2007. 6

[21] Edouard Leurent. An Environment for Autonomous Driving Decision-Making, 2018. URL https://github.com/Farama-Foundation/HighwayEnv. 4

[22] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*, 2020. 7

[23] Boyi Li, Yue Wang, Jiageng Mao, Boris Ivanovic, Sushant Veer, Karen Leung, and Marco Pavone. Driving Everywhere with Large Language Model Policy Adaptation. In *CVPR*, 2024. 2

[24] Xin Li, Yeqi Bai, Pinlong Cai, Licheng Wen, Daocheng Fu, Bo Zhang, Xuemeng Yang, Xinyu Cai, Tao Ma, Jianfei Guo, Xing Gao, Min Dou, Botian Shi, Yong Liu, Liang He, and Yu Qiao. Towards Knowledge-driven Autonomous Driving. *arXiv*, 2023. 1

[25] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. In *ICRA*, 2023. 1, 2

[26] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models. In *NeurIPS*, 2023. 2

[27] Jiageng Mao, Yuxi Qian, Junjie Ye, Hang Zhao, and Yue Wang. GPT-Driver: Learning to Drive with GPT. *NeurIPS Workshop on Foundation Models for Decision Making*, 2023. 2

[28] Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. A Language Agent for Autonomous Driving. *arXiv*, 2023. 2

[29] Ana-Maria Marcu, Long Chen, Jan Hünermann, Alice Karnsund, Benoit Hanotte, Prajwal Chidananda, Saurabh Nair, Vijay Badrinarayanan, Alex Kendall, Jamie Shotton, and Oleg Sinavski. LingoQA: Video Question Answering for Autonomous Driving. *arXiv*, 2023. 2

[30] McKinsey Center for Future Mobility. Autonomous driving's future: convenient and connected, 2023. URL https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected. 1

[31] Ming Nie, Renyuan Peng, Chunwei Wang, Xinyue Cai, Jianhua Han, Hang Xu, and Li Zhang. Reason2Drive: Towards Interpretable and Chain-based Reasoning for Autonomous Driving. *arXiv*, 2023. 2

[32] OpenAI. GPT-4-Turbo with 128K context and lower prices, the new Assistants API, GPT-4 Turbo with Vision, DALL·E 3 API, and more, 2023. URL https://openai.com/blog/new-models-and-developer-products-announced-at-devday. 6

[33] OpenAI. GPT-4 Technical Report. *arXiv*, 2023. 6

[34] OpenAI. Introducing ChatGPT, 2023. URL https://openai.com/blog/chatgpt. 6

[35] Chenbin Pan, Burhaneddin Yaman, Tommaso Nesti, Abhirup Mallik, Alessandro G. Allievi, Senem Velipasalar, and Liu Ren. VLP: Vision Language Planning for Autonomous Driving. In *CVPR*, 2024. 2

[36] Hao Sha, Yao Mu, Yuxuan Jiang, Li Chen, Chenfeng Xu, Ping Luo, Shengbo Eben Li, Masayoshi Tomizuka, Wei Zhan, and Mingyu Ding. LanguageMPC: Large Language Models as Decision Makers for Autonomous Driving. *arXiv*, 2023. 2

[37] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a Formal Model of Safe and Scalable Self-driving Cars. *arXiv*, 2017. 3

[38] Hao Shao, Letian Wang, Ruobing Chen, Steven L. Waslander, Hongsheng Li, and Yu Liu. ReasonNet: End-to-End Driving with Temporal and Global Reasoning. In *CVPR*. arXiv, 2023. 1

[39] Hao Shao, Yuxuan Hu, Letian Wang, Steven L. Waslander, Yu Liu, and Hongsheng Li. LMDrive: Closed-Loop End-to-End Driving with Large Language Models. In *CVPR*, 2024. 2

[40] Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Ping Luo, Andreas Geiger, and Hongyang Li. DriveLM: Driving with Graph Visual Question Answering. *arXiv*, 2023. 2

[41] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *ICRA*, 2023. 2

[42] Jiankai Sun, Chuanyang Zheng, Enze Xie, Zhengying Liu, Ruihang Chu, Jianing Qiu, Jiaqi Xu, Mingyu Ding, Hongyang Li, Mengzhe Geng, Yue Wu, Wenhai Wang, Junsong Chen, Zhangyue Yin, Xiaozhe Ren, Jie Fu, Junxian He, Wu Yuan, Qi Liu, Xihui Liu, Yu Li, Hao Dong, Yu Cheng, Ming Zhang, Pheng Ann Heng, Jifeng Dai, Ping Luo, Jingdong Wang, Ji-Rong Wen, Xipeng Qiu, Yike Guo, Hui Xiong, Qun Liu, and Zhenguo Li. A Survey of Reasoning with Foundation Models. *arXiv*, 2024. 1

[43] Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual Inference via Python Execution for Reasoning. In *ICCV*, 2023. 2

[44] Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Chenxu Hu, Yang Wang, Kun Zhan, Peng Jia, Xianpeng Lang, and Hang Zhao. DriveVLM: The Convergence of Autonomous Driving and Large Vision-Language Models. *arXiv*, 2024. 2

[45] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Vik-

tor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv*, 2023. 6

[46] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 2000. 6

[47] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv*, 2023. 7

[48] Qifan Wang, Jingang Wang, Xiaojun Quan, Fuli Feng, Zenglin Xu, Shaoliang Nie, Sinong Wang, Madian Khabsa, Hamed Firooz, and Dongfang Liu. MUSTIE: Multimodal Structural Transformer for Web Information Extraction. In *ACL*, 2023. 2

[49] Wenhai Wang, Jiangwei Xie, ChuanYang Hu, Haoming Zou, Jianan Fan, Wenwen Tong, Yang Wen, Silei Wu, Hanming Deng, Zhiqi Li, Hao Tian, Lewei Lu, Xizhou Zhu, Xiaogang Wang, Yu Qiao, and Jifeng Dai. DriveMLM: Aligning Multi-Modal Large Language Models with Behavioral Planning States for Autonomous Driving. *arXiv*, 2023. 2

[50] Xingyao Wang, Sha Li, and Heng Ji. Code4Struct: Code Generation for Few-Shot Event Structure Prediction. In *ACL*, 2023. 2

[51] Xingyao Wang, Hao Peng, Reyhaneh Jabbarvand, and Heng Ji. LeTI: Learning to Generate from Textual Interactions. *arXiv*, 2023. 2

[52] Yixuan Wang, Ruochen Jiao, Chengtian Lang, Sinong Simon Zhan, Chao Huang, Zhaoran Wang, Zhuoran Yang, and Qi Zhu. Empowering Autonomous Driving with Large Language Models: A Safety Perspective. In *ICLR Workshop on LLM Agents*, 2024. 8

[53] Wayve Technologies Ltd. LINGO-1: Exploring Natural Language for Autonomous Driving, 2023. URL https://wayve.ai/thinking/lingo-natural-language-autonomous-driving/. 2

[54] 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. In *NeurIPS*, 2022. 2, 6

[55] Licheng Wen, Xuemeng Yang, Daocheng Fu, Xiaofeng Wang, Pinlong Cai, Xin Li, Tao Ma, Yingxuan Li, Linran Xu, Dengke Shang, Zheng Zhu, Shaoyan Sun, Yeqi Bai, Xinyu Cai, Min Dou, Shuanglu Hu, Botian Shi, and Yu Qiao. On the Road with GPT-4V(ision): Early Explorations of Visual-Language Model on Autonomous Driving. *arXiv*, 2023. 2

[56] Licheng Wen, Daocheng Fu, Xin Li, Xinyu Cai, Tao Ma, Pinlong Cai, Min Dou, Botian Shi, Liang He, and Yu Qiao. DiLu: A Knowledge-Driven Approach to Autonomous Driving with Large Language Models. In *ICLR*, 2024. 2

[57] Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee K. Wong, Zhenguo Li, and Hengshuang Zhao. DriveGPT4: Interpretable End-to-end Autonomous Driving via Large Language Model. *arXiv*, 2023. 2

[58] Ke Yang, Jiateng Liu, John Wu, Chaoqi Yang, Yi R. Fung, Sha Li, Zixuan Huang, Xu Cao, Xingyao Wang, Yiquan Wang, Heng Ji, and Chengxiang Zhai. If LLM Is the Wizard, Then Code Is the Wand: A Survey on How Code Empowers Large Language Models to Serve as Intelligent Agents. *ICLR Workshop on LLM Agents*, 2024. 2

[59] Yi Yang, Qingwen Zhang, Ci Li, Daniel Simões Marta, Nazre Batool, and John Folkesson. Human-Centric Autonomous Systems With LLMs for User Command Reasoning. In *WACVW*, 2024. 2

[60] Zhenjie Yang, Xiaosong Jia, Hongyang Li, and Junchi Yan. LLM4Drive: A Survey of Large Language Models for Autonomous Driving. *arXiv*, 2023. 1

[61] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*, 2023. 1

[62] Wenqian Ye, Guangtao Zheng, Xu Cao, Yunsheng Ma, Xia Hu, and Aidong Zhang. Spurious Correlations in Machine Learning: A Survey. *arXiv*, 2024. 8

[63] Jianhao Yuan, Shuyang Sun, Daniel Omeiza, Bo Zhao, Paul Newman, Lars Kunze, and Matthew Gadd. RAG-Driver: Generalisable Driving Explanations with Retrieval-Augmented In-Context Learning in Multi-Modal Large Language Model. *arXiv*, 2024. 2

[64] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. Large Language Models Meet NL2Code: A Survey. In *ACL*, 2023. 2

[65] Xingcheng Zhou, Mingyu Liu, Bare Luka Zagar, Ekim Yurtsever, and Alois C. Knoll. Vision Language Models in Autonomous Driving and Intelligent Transportation Systems. *arXiv*, 2023. 1

[66] Yunsong Zhou, Linyan Huang, Qingwen Bu, Jia Zeng, Tianyu Li, Hang Qiu, Hongzi Zhu, Minyi Guo, Yu Qiao, and Hongyang Li. Embodied Understanding of Driving Scenarios. *arXiv*, 2024. 2