# DRIVINGBENCH: CAN VISION-LANGUAGE MODELS DRIVE A TOYOTA COROLLA?

**Aditya Ramabadran*** **Simon Mahns*** **Tobias Gessler***

team@drivingbench.com

## ABSTRACT

Frontier models excel at many digital benchmarks, yet their ability to drive a real car, an everyday human skill, remains largely untested. We present DrivingBench, to our knowledge the first benchmark where general-purpose vision-language models must drive a real car. Through three tools, the models see camera frames from a Toyota Corolla and directly command its steering and velocity around a parking lot cone course at low speeds. The car may continue moving while the model thinks and new commands replace the currently running one, so inference latency is part of the task, testing the models' abilities to observe, act, monitor, recover, and complete a long-horizon objective under such constraints. We benchmark GPT-6 Astra, Claude Fable 5.1, GPT-5.6 Sol, and Grok 4.6 in vendor-native harnesses (Codex, Claude Code, Cursor) with up to three attempts each in one conversation; Astra is the only model to finish the course, on its second attempt, with no other attempt passing 50% of the course. Two of the four models improved materially across attempts with retained context. We also detail the design principles behind our action interface, and show how the tool output format and the framing of the task combined to determine whether models would drive at all or refuse. We release our harness, prompts, course map, and traces with video and telemetry for reproducibility.

Project page: drivingbench.com Code: GitHub X: @drivingbench

## 1 INTRODUCTION

AI has made rapid progress across a wide range of digital tasks. Benchmarks built to measure frontier models often saturate within a few years of release (Kiela et al., 2021), prompting ever harder ones, from broad knowledge and expert-level science questions to real software engineering and research-level mathematics (Hendrycks et al., 2021; Rein et al., 2024; Jimenez et al., 2024; Phan et al., 2025; Glazer et al., 2024). Yet, as observed by Moravec (1988), skills that come effortlessly to people like perception and acting in the physical world are some of the hardest to reproduce in AI. Driving is a clear example: sixteen-year-olds can steer a car around a cone course in an empty parking lot in well under a minute.

Frontier models have recently started acquiring the ingredients a task like this needs: advanced abilities in vision, spatial/3D reasoning, planning, tool use, and the ability to learn from their past attempts in-context. Through vendor applications such as Codex, Claude Code, and Cursor, they are increasingly deployed as agents that act and use tools, rather than chatbots that only answer questions. This raises a simple question that has not yet been measured: can a *frontier model* (defined here as a general-purpose vision-language model (VLM), used unmodified through such agent applications), with likely no driving-specific training, drive a real car?

We introduce DrivingBench to answer this. There are three properties that distinguish DrivingBench from previous evaluations and benchmarks. First, the model itself is the driver. It sees camera images and chooses steering and speed commands itself, with nothing between it and the car except for openpilot's low-level actuation (comma.ai, 2026). Second, the evaluation runs fully in the real

*Equal contribution.

world and is closed-loop: the controls aren't familiar to the agent and have to be calibrated through feedback and in-context learning, and, unlike robotic pick-place tasks, mistakes can't simply be undone or retried in the same attempt. Third, the car continues executing an active command while the model chooses its next action. Each new command given replaces the currently running one, so latency becomes an integral part of the task.

Language models have been put in real vehicles before, but either as models trained for driving or as a layer on top of a conventional autonomy stack. Talk2Drive (Cui et al., 2024) turns verbal requests into parameters for a pre-existing driving stack, DriveVLM (Tian et al., 2025) fine-tunes a VLM to produce driving decisions and coarse trajectories that a conventional planner refines, and LINGO-2 (Wayve, 2024) is a vision-language-action model trained on driving data and tested on public roads. But in DrivingBench, the model itself is the driver. To our knowledge, it's the first evaluation in which frontier models perceive a real car's surroundings from cameras, and *directly* command its steering/speed in closed-loop.

Our setup involves a 2022 Toyota Corolla, to which we retrofitted a comma four running a modified version of the openpilot software (comma.ai, 2026). The model drives it via three Model Context Protocol tools (Anthropic, 2024): `observe`, `set_motion` and `stop_now`. Figure 1 shows an overview of the models' inputs and controls of the car. Every model gets up to three attempts at a fixed 127 m cone course in a single continuous chat, with a generic reflection prompt after each failed attempt. We evaluate GPT-6 Astra and GPT-5.6 Sol in Codex, Claude Fable 5.1 in Claude Code, and Grok 4.6 in Cursor. GPT-6 Astra completed the course on its second attempt, reaching the finish zone 4 minutes and 44 seconds after its first command. Claude Fable 5.1 reached 45% of the course on its third attempt, while Grok 4.6 and GPT-5.6 Sol failed to pass the first corner. Common failure modes included misreading which side of a cone line the lane is on (perception), or acting on stale/old observations and committing prematurely to plans (planning, latency). Two out of the four models improved across attempts. The other two accurately diagnosed their mistakes in their reflections, but did not materially change their driving.




Figure 1: **A frontier model driving a car through a fixed cone course.** (1) Camera frame returned by `observe()`. (2) the model calls the MCP to control the car. (3) evaluation trajectory through the fixed cone course.

General-purpose models are beginning to show the ability to act in the real physical world out-of-the-box. To measure their progress on those real-world tasks we introduce DrivingBench, an open-source evaluation harness that connects MCP-capable agent applications to real cars. It includes three simple tools and layered safety limitations (Section 3). We describe the design principles and ideas behind our action interface, which is what the model uses to send commands and receive data while the car moves (Section 3.1), and show that the combination of how we frame the task and what the tools report to the model helped determine whether models would drive or refuse to do so (Section 6). We release all traces, commands, transcripts, and appropriately-blurred videos. While our results show that general purpose models are now capable of completing real-world tasks such as driving at a very basic level, there are numerous failures in perception, planning, latency, and a gap between knowing and doing in in-context learning (Paglieri et al., 2025; Schmied et al., 2026). Nevertheless, these new model capabilities raise questions of safety, alignment, and ethics, that we return to in Section 8. We believe that this benchmark pioneers an important new paradigm for evaluating frontier models on real-world tasks. AI model capabilities have reached an inflection point where these physical tasks become feasible, and rigorous evaluation will be essential to measure model progress.

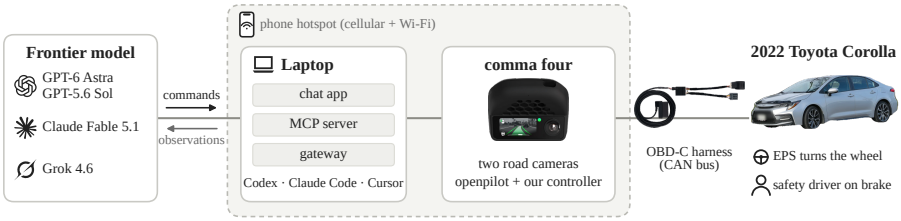



Figure 2: **DrivingBench System.** The agent applications run on a laptop (Codex, Claude Code, Cursor) and have access to our DrivingBench MCP server. The laptop used for evaluation and the comma four computer are both connected to the same phone hotspot. The comma device runs a modified version of the openpilot software, allowing the MCP to observe the front-facing camera and issue steering commands to the car.

## 2 Related Work

Driving evaluations of language models have mostly been open-loop question answering over recorded scenes (Sima et al., 2024), or simulated closed-loop driving (e.g. in the CARLA simulator) with models trained for driving (Shao et al., 2024) or small general-purpose VLMs (Jia et al., 2026). The closest simulator experiments to our setup have evaluated LLMs at driving through code and APIs (Ma et al., 2024), with memory and reflection across episodes (Wen et al., 2024) just like our protocol does in a car.

General-purpose multimodal agents are benchmarked in simulation where they appear strong at high-level tasks but relatively weak at lower level control (Yang et al., 2025), as well as in real software environments where they are graded by execution (Xie et al., 2024).

In robotics, language models write policy code (Liang et al., 2023), choose among pre-learned skills (Ichter et al., 2023), or are trained on robot data (such as vision-language-action models, or VLAs) (Zitkovich et al., 2023). Untuned vision-language models can produce continuous actions on real robots, but the robot waits for every query (Nasiriany et al., 2024; Menon et al., 2026). By contrast, our car keeps moving while a command is still running, which is closer to policies that must already plan future actions while executing the current actions (Black et al., 2025). Our reflection prompt follows verbal self-reflection between trials (Shinn et al., 2023). The knowing-doing gap we observe (Section 5.4) has been reported previously in games and bandits (Paglieri et al., 2025; Schmied et al., 2026).

Robots controlled by LLMs can be coerced into unsafe physical actions (Robey et al., 2025; Zhang et al., 2025; Sun et al., 2026), and models often can show awareness of when they are being evaluated (Needham et al., 2025; Laine et al., 2024). This might help partly explain why the framing of the task and tool output formats affected whether some models would drive (Section 6). The same model can behave in opposite ways depending on the situation: GPT-6 Astra refused only 2 of 100 harmful robot-arm instructions in RoboHarm when simply asked to carry them out (Sun et al., 2026), yet during DrivingBench evaluations it refused to drive in an empty parking lot until we reframed the task and iterated heavily on our prompt.

## 3 DrivingBench System

To facilitate DrivingBench evaluations we provide a reproducible system to connect AI models to a real car. Figure 2 provides an overview of the driving system. To perform the evaluations we use a 2022 Toyota Corolla. We install a comma four[1] device, an aftermarket autonomy kit, that allows old cars to be retrofitted with self-driving capabilities. The comma is a computer mounted on the windshield that connects to the car via an OBD-C cable to the CAN bus and allows us to send commands to the car to control steering, acceleration and brakes. Additionally, the comma provides a forward-facing and a wide-angle camera view ($1344\times760$ images). The comma device

[1] https://www.comma.ai/shop/comma-four

and a laptop, running the AI model that is being evaluated, are then both connected to the same wireless network.

**Driving Harness.** We provide an open-source implementation of our driving harness[2] including the car controller and MCP server (Anthropic, 2024). The model can take control of the car via the MCP server and has the following tools available:

**`observe()`** camera frames (narrow + wide), speed, steering, time left on command

**`set_motion(direction, steering_percent, speed_mps, duration_s, reason)`**

| | |
|---|---|
| `direction` | `left` \| `right` \| `straight` |
| `steering_percent` | 0–100 % of the maximum steering-wheel angle (a parameter we set to $180^\circ$) |
| `speed_mps` | target speed, which we set to 0.5–3.5 m/s |
| `duration_s` | 5–60 s, after which the car brakes |
| `reason` | freeform text, describing the intention/reasoning behind the command for logging |

**`stop_now(reason)`** brake immediately (does not end the session)

**Driving Controller.** The MCP server then issues those commands to the car via our driving controller. The driving controller runs on the comma computer and is built on a modified version of the openpilot software (comma.ai, 2026). The controller can receive motion commands from the MCP, translates them into steering-wheel angles and submits those to the car via openpilot. openpilot's vehicle model converts this angle into a curvature target, which its steering controller and the car interface turn into bounded torque steering, while a separate loop tracks the requested speed.

**Operator Platform.** Alongside the MCP server we also provide an operator platform which can be used to manually control the car as well as change configuration details such as camera exposure or saturation. The operator portal also visualizes the live camera views and all the tool calls the model performs in real-time, allowing the operator to inspect the models' actions. Additionally, the car's traces including telemetry and tool calls are recorded and can be used to analyze and document evaluations. An overview of our trace viewer which we use to visualize these recorded trajectories can be found in Figure 6.

**Model Refusals.** While developing DrivingBench, some models (GPT-6 Astra in particular) often refused to drive the real car. Section 6 describes our analysis on what triggered the refusals, and how we were eventually able to resolve them. With our final tool interface and prompt/framing (Appendix A), every model drove on every attempt.

### 3.1 Action interface design

The interface above came after several rounds of iteration on earlier designs. Our initial design allowed models to submit geometric paths, via one of these three formats: metric waypoints, pixel/coordinate paths in the camera image, or chains of curvature segments. The harness and code layer would compile and track these paths, along with owning admission checks, turn shaping, and endpoint braking. Later, we had a version which replaced paths with timed curvature setpoints. Our final version described above replaced all of this with a steering percent, speed, and duration (Table 1). Each change was inspired by a failure mode of the previous design. Although we don't have controlled comparisons, our development process yielded the following general principles.

**Explicitly expose latency rather than absorbing it.** Paths are planned relative to the frame that the model saw, but the car can continue moving while the model thinks or between commands. With paths, the motion of the car would (silently) consume the initial part of the route. For example, in one development test run, the car moved 2.4 m while the model planned, and our harness then rejected a valid turn because the car's steering was no longer close to the chosen path's entry. We then tried re-anchoring paths to the car's live trajectory/position. This removed the rejection, but it just hid the delay further. In our final design, commands take effect when accepted, and all tool responses come with both a timestamp and the image's age. The car continues moving with a running command while the model thinks, but latency ended up becoming an explicit and visible part of the task (Section 5.2) instead of a hidden or absorbed source of errors.

[2] https://github.com/aditya-ramabadran/drivingbench_harness_v1

Table 1: **Initial (work-in-progress) and final action interfaces.**

| | **Initial interfaces (paths)** | **Final interface** |
|---|---|---|
| Model output | route geometry (waypoints, image-pixel paths or curvature segments) | steering %, speed, duration |
| Tools | `observe`, `submit_path`, `wait_held`, `final_stop` | `observe`, `set_motion`, `stop_now` |
| Harness owns | path tracking, accepting commands (as valid), turn shaping, endpoint braking | native openpilot limits only |
| Latency | absorbed: the car moves along the route while the model plans, no explicit exposure to the models | explicit: commands only start when accepted, and every response is timestamped |
| New command behavior | gets merged with the rest of the old existing/active path | replaces the running command, and expiration brakes |
| Infeasible/invalid request | rejected (explicitly) | clamped by default, only speeds above the ceiling are rejected |

**Command in units that make sense to the model.** Earlier designs used curvature in $m^{-1}$ and distances in meters as command inputs, but these aren't quantities that can be easily read off a camera image, and the models weren't calibrated enough in absolute distance to use them properly. Steering percents aren't more physical, but since `observe` reports the current measured steering on the same scale, the model can easily compare what it asked for with what the car did over time (since requested steering percents take time to ramp up on the steering wheel, and may not ramp up fully based on the speed and conditions). The models took advantage of this: Sol noted that "two seconds into that command, measured steering was still approximately 1%" (Section 5), and in-context learning and improvement across attempts (Section 5.4) depends a lot on this sort of self-calibration.

**Reduce rejection of commands, bound or clamp instead.** Our earlier path harness would check every command submission against a turning envelope and reject ones that were infeasible. Models would then have to guess parameters, were refused repeatedly, and would guess again. Each rejection would cost a full model turn, while the car would either keep moving or wait stationary. Our final harness instead clamps commands to native limits, and only rejects speeds above a shared ceiling. Rejected commands also leave the previous one running. It was important to reduce rejections and refusals as much as possible, both on the model (Section 6) and harness side, and increase the proportion of turns and times that models were moving the car.

**Replace commands actively rather than queuing them.** In our final design, every new command replaces the running one, and a command expiring just means braking the car. Under the earlier path interface, driving continuously (replacing a path before it ran out) required re-expressing the rest of the old route in the car's current frame, and keeping a valid plan or path for all times. In a profiling run with repeated 400 ms dispatch delays, the controller lacked a valid plan for 2.7% of control ticks, since any single such delay was enough to expire the plan. When we moved toward pure replacement instead, commands that come too late would just let the previous one expire into a stop as needed before taking over.

**Reduce the number of parameters.** Initial interfaces offered multiple path formats and tuning options to models, and models spent reasoning on choosing parameters rather than on the road, and nevertheless ran into our harness's curvature limit often. Our final `set_motion` command has just four simple control arguments, and the whole command path from the MCP tool to controller is only around a thousand lines of Python code.

**Ask for reasoning or intent.** Closed-source proprietary models like the frontier models we tested sometimes expose terse summaries of reasoning, but never raw reasoning traces, and this wasn't enough to analyze models' intents and reasoning behind their actions. In our final design, every command has a short free-form field called `reason` for models. It helps us record what the model believed when it acted, and why, which made categorizing failure modes much easier and made the analysis in Section 5 possible.

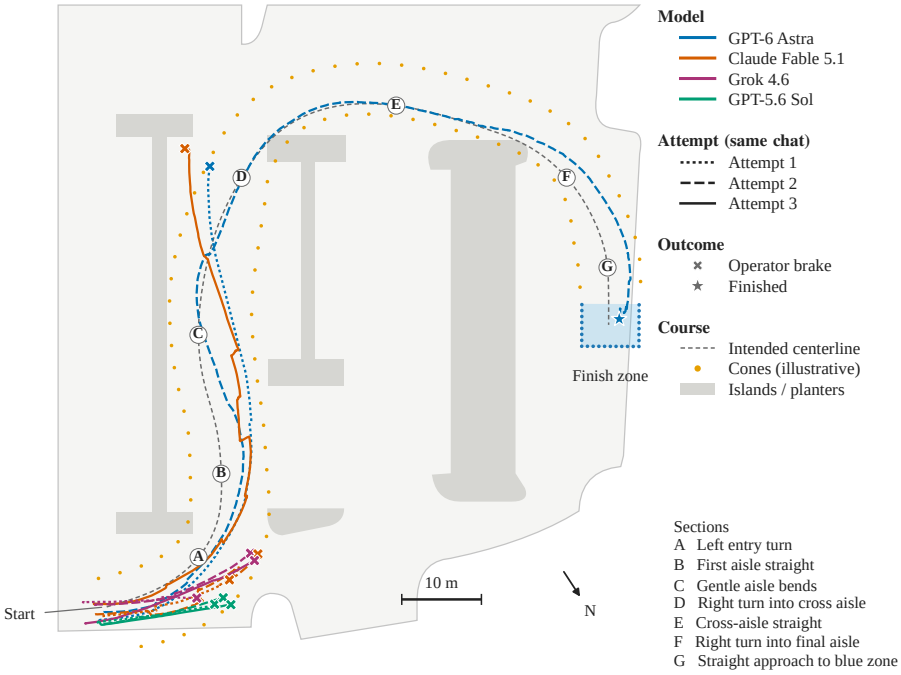



Figure 3: **GPS trajectories of all 11 attempts on the cone course.** Color identifies the model and line style the attempt within one chat. Markers show where the safety driver braked or where the car finished. The letters mark seven course sections (turns, bends, straights). 8 of the 11 attempts ended before section B, failing the first turn. The centerline was inferred from visible cones. Cone positions are illustrative (estimated from drone footage and satellite maps), rather than precisely surveyed. The reference used for progress/benchmark percents (Section 4.3) is the centerline up to the finish zone, which is 127 m long.

# 4 Evaluation Setting

## 4.1 Course

The evaluation task is to have each model/harness pair drive through a cone course and park in its finish zone (Figure 3). Multicolored mini-cones and tall orange cones mark a corridor that runs about 47 m up the main aisle, 27 m across the lot and 18 m down the final aisle, into a finishing $7 \times 5$ m zone of blue mini-cones. The route combines a left turn, two right turns, straights and gentle bends. Turn radius design was guided by openpilot's steering limits.

## 4.2 Models and Trial setup

We evaluate four models, each in an agent harness at medium reasoning effort: GPT-6 Astra and GPT-5.6 Sol in Codex (CLI 0.154.0-alpha.6.2), Claude Fable 5.1 in Claude Code (2.1.274), and Grok 4.6 in Cursor. Each harness adds its own system prompt, tool interface and latency, so we compare model–harness systems rather than models. Results thus reflect models as deployed with their native harness, and separating the model from harness is left to future work. The four trials ran consecutively in the order Astra, Sol, Fable, Grok.

A trial is a one-time chat session. It begins with a fixed prompt (Appendix A) that states the objective (arrive at the parking area marked by blue cones), the action semantics, general driving guidance, and informs the model it is scored first on how far it gets without leaving the course, followed by time to completion. A trial allows up to three attempts and terminates at the first success. After a failed attempt the operator sends a fixed reflection prompt (Shinn et al., 2023), followed by a fixed continuation prompt, in the same session (both in Appendix A). Attempts within a trial therefore share context and are not independent.

Table 2: **Results.** Progress (%) per attempt; bold marks the first attempt that reached the best score for that model. Finish time gives the time between the first accepted command and the end of the last engagement. Tokens and cost are summed over all attempts (including reflections) at the public list prices as of September 2026. All models here ran with medium reasoning effort.

| | | Progress per attempt (%) | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Model | Harness | 1 | 2 | 3 | Best (%) | Finish time | Tokens | Cost ($) |
| GPT-6 Astra | Codex | 49 | **100** | – | 100 | 5:22 | 7.8M | 9.75 |
| Claude Fable 5.1 | Claude Code | 9 | 10 | **45** | 45 | DNF | 3.6M | 3.95 |
| Grok 4.6 | Cursor | 8 | **11** | 10 | 11 | DNF | 1.0M | 0.65 |
| GPT-5.6 Sol | Codex | **6** | 6 | 6 | 6 | DNF | 1.6M | 1.05 |

### 4.3 Metrics

The primary metric is *progress*, defined as the furthest point along the course centerline that the car reached while within 4 m of it, as a share of the centerline's 127 m up to the finish zone, so 100% means the car reached the finish zone. We compute it from the comma's GPS (median reported accuracy 0.9 m, maximum 1.2 m) between the first accepted command and the end of the attempt's last engagement. Progress pauses while the car is more than 4 m away, and each GPS fix can only advance it, by at most 20 m, so stops and wrong-way driving add nothing and the car cannot gain progress by cutting across where the course folds back. Rankings are unchanged for bands of 4–6 m; with 3 m, Astra's first attempt, which swung up to 5.4 m wide before recovering, would score 17% instead of 49%. Over the same window, we additionally report *time*, *distance* (integrated GPS speed) and the number of accepted `set_motion` *commands*, as well as *tokens* and *cost* at public list prices on the day of the trial, from each harness's usage records, including the reflection after each attempt. This metric of progress is imperfect and may not truly capture the "percent" of driving ability each model shows, but we believe it is a viable choice for attempting to encapsulate our benchmark results into one number.

### 4.4 Operator Protocol and Safety

An attentive driver sits in the driver's seat with a foot constantly over the brake throughout evaluations (Figure 2). Before each attempt, the car is driven back to the same marked start position, and our operator always made sure to align the car and its wheels to a fixed line in the lot, fixing both the car's position and heading for each trial. The operator engages openpilot and presses the RES button once the model's first command is accepted, since openpilot does not move the car from a standstill without this button being pushed. The car began moving a median of 3.0 s after that command. The operator hits the brakes when the car leaves the course, or is about to hit a curb, island or other obstacle, which ends the attempt. All 10 failed attempts ended this way (Appendix D); the successful attempt instead ended with the model's own `stop_now` inside the finish zone, after which the operator disengaged.

The operator sent two messages beyond the fixed prompts, both visible in the released transcripts and described in Appendix B.1: "Continue" once to Astra, and "Stop" to end Grok's attempts.

## 5 Results

In this section, we discuss the results of our evaluations and experiments. This includes 11 attempts over 4 models (Astra has 2: it finished on attempt 2, so no third attempt). We include per-attempt statistics (e.g. distance, moving share, replacements, gaps, peak speed) in Table 4, Appendix D.

### 5.1 Course completion

Table 2 shows the overall results. GPT-6 Astra in attempt 1 reached 49% before drifting over the left cone line and toward the island, at section D (Figure 3). Its second attempt, however, finished the entire course and parked in the blue finish zone. It entered the finish zone 4:44 after its first accepted

command and called `stop_now` there at 5:05; the engagement ended at 5:22 (322 seconds, the finish time we report), after 134.7 meters and 24 commands.

Claude Fable 5.1 improved from 9% to 10% to 45% over the course of its three attempts. Its first two attempts both crossed the cone line at A, but its third attempt surpassed this and drove the long aisle (B, C); similarly to Astra's first attempt, it then drifted too far left around D.

Grok 4.6 in its three attempts achieved 8%, 11%, and 10%; and GPT-5.6 Sol achieved 6% in all three attempts. All six of the attempts between these two models ended at A, within the first $\sim$15 m.

Figure 7b (Appendix D) hints at some differences in command strategy between the models. Astra progresses steadily (at $\approx$0.3% of the course per second), while Fable progresses in steps with very long, flat stretches. These are periods where its previous motion command ended, and the car waited stationary while Fable was reasoning, until it gave its next command.

### 5.2 Latency is part of the task

Our benchmark has the unique feature that the car keeps moving in the real world while the model thinks, so long as it has an ongoing motion command. Therefore, latency and inference speed and models' choices on how long to reason between commands all become part of the evaluation.

The median time from the previous tool call to a `set_motion` command for each model was: Astra 4.9 s, Sol 5.4 s, Grok 12.5 s, Fable 22.2 s (Figure 8a, Appendix D). Notably, models must also act on stale images. The car had driven a median of $\approx$4 m (Astra) between when the previous frame the model saw was captured, and when its motion command was accepted (Figure 8b).

Our tools also allow for an ongoing/running motion command to be replaced by a new command or tool call from the model. This is as opposed to queuing motion commands or waiting for a running command to complete its duration. This was used only by Astra (4 of 30) and Sol (1 of 6); Fable (0 of 12) and Grok (0 of 5) never replaced a running command.

Our setup means that expiry can be costly. The EPS only turns the wheel once the car rolls, so from rest, the wheel needs a median of 4.0 s to reach 90% of a steering change, versus just 1.9 s when rolling (Figure 9b). A stop-and-go strategy (such as Fable's) thus costs steering ability, especially at the turns. Appendix B.2 gives three examples of how exactly this played out.

### 5.3 Failure modes

Only one attempt succeeded, and the other ten attempts ended unsuccessfully (with an operator brake). Figure 4 shows one such failed attempt per model. Many of the causes we can infer for these failures fall into three broad groups (and latency, a fourth, is covered in Section 5.2), and generally align with the models' own reflections. Of course, any given failure can be assigned to multiple of these causes (or others), and these failures can compound each other (e.g. worse latency can make a model use a stale frame and overcommit to a premature plan).

**Reading the cone boundary.** This is a perception failure that affected 8 of the 11 attempts, at section A (including every Sol and Grok attempt, and the first two Fable attempts). Despite being able to see both cone lines across the frames, the model makes incorrect assumptions about which side of the first diagonal cone line the intended lane is on.

- Fable (attempt 2, turn 44): "I picked the wrong side of the boundary again. The diagonal cone line was the lane's left edge, not its right edge."
- Sol decided to use cone color to determine the side. Its final command in Figure 4 is explained as "centered between the near green-left and red-right cones"; its reflection (attempt 2, turn 51) notes that "The decisive error was assuming that cone color identified boundary side."
- Grok attempt 1 drove straight into the cone gate (turn 13): "The car is wider than the camera makes it look." This is despite our prompt warning the models that "the vehicle you are controlling is also 3D and is also wider than it might seem from the camera images."

**Planning and premature commitment.** This failure mode involves the model reading the scene roughly correctly, but then committing to a plan of some sort (e.g. a speed, or a gentle turn, or

a direction) before it had enough information to be sure it was right, or without leaving room for change/error/leeway (e.g. the car's turning radius). This includes the two furthest failed attempts (Astra attempt 1 and Fable attempt 3).

- Astra's first attempt sped up too early while it was still straight and didn't leave time to turn right into section D: "I declared the car aligned too early. [...] I straightened and increased the requested speed to 1.5 m/s." (turn 51). Its final command was a 100% right turn into section D, but it came too late to make it (as shown in Figure 4, top).
- Fable's third attempt drove all the way through section C well, but then it started the right turn at D with only a "gentle 30% right to follow the curve" at 1.2 m/s (turn 82). This was far too gentle and left too little room, and soon after, the operator had to brake with the car pointed toward the top-left island (Figure 3).

**Vehicle/hardware limits.** The rate and the range at which openpilot can turn the steering wheel via the EPS motor are both limited. The tightest turning radius possible is thus limited (from our data, $\approx$16 m at the typical $\approx$130° wheel angle), and turns also take time to build up especially when requested from rest. This means that corrections that start too late can't possibly be recovered when near islands or curbs. Appendix C describes some of the mechanisms we measured behind these limits. The models noticed some of these limits, but did not always interpret them correctly or apply their conclusions in subsequent attempts.

- Sol's first attempt (turn 21) included: "Steering built slowly; two seconds into that command, measured steering was still approximately 1%, so the car continued nearly straight." And by its last observation, "Measured steering had reached only about 17%."
- Fable noticed the car turned more than it expected in its second attempt (turn 44): "I misjudged turn radius again. The car rotates far more than I expected at 100% over eight seconds," and planned to "Use 30% steering as the default."
- The turning radius limitations had some contribution to Fable's failure in its third attempt (Figure 4, second row) as its 30% right turn at D, sent 20 s after its last frame, was far too gentle to make it.

### 5.4 In-context learning across attempts

Two out of the four models, GPT-6 Astra (49 $\rightarrow$ 100%) and Claude Fable 5.1 (9 $\rightarrow$ 45%), improved materially across their attempts. Grok 4.6 and GPT-5.6 Sol did not really improve and stayed unable to clear the first turn (Table 2).

The reflections were largely accurate. The models even restated the fixes at the beginning of their next attempt. The issue, however, was actually turning it into a different or improved control procedure. This "knowing-doing" gap for LLM agents (in bandits and games) has been reported before (Paglieri et al., 2025; Schmied et al., 2026), but is now seen here in this real-world closed loop.

- Grok acted on its first reflection: after its first attempt, it resolved to commit to the left turn needed (turn 13), and in attempt 2 it immediately started turning left from rest. In its reflection after its second attempt (turn 28), it said: "Overlap every command. Replace while several seconds remain." In its third attempt, it announced "short overlapping moves" (turn 30). However, all three of its commands arrived after the previous one had already expired and the car had come to rest (0 of 2 motion command replacements, Table 4).
- Sol opened its third attempt by declaring that it will "treat cone colors as irrelevant ... I'll follow the continuous multicolored cone line as the left boundary" (turn 53). However, it then sent two straight commands and ended up in the same place it ended previously (6%).
- Fable's improvement in its last attempt was one of perception (rather than steering/control). Starting its third attempt, Fable declared "This time I keep the diagonal cone line on my left and run along the parking stalls" (turn 46) and sent four 100% left commands.
- Astra's improvement is even more clear. In its first reflection (attempt 1, turn 51), it proposed "shorter travel increments at about 0.5–0.8 m/s" near bends and islands. Its second attempt then only sent commands for 0.5–0.8 m/s (attempt 1: 0.7–1.5), and observed 33 times instead of 13, and finished the course.

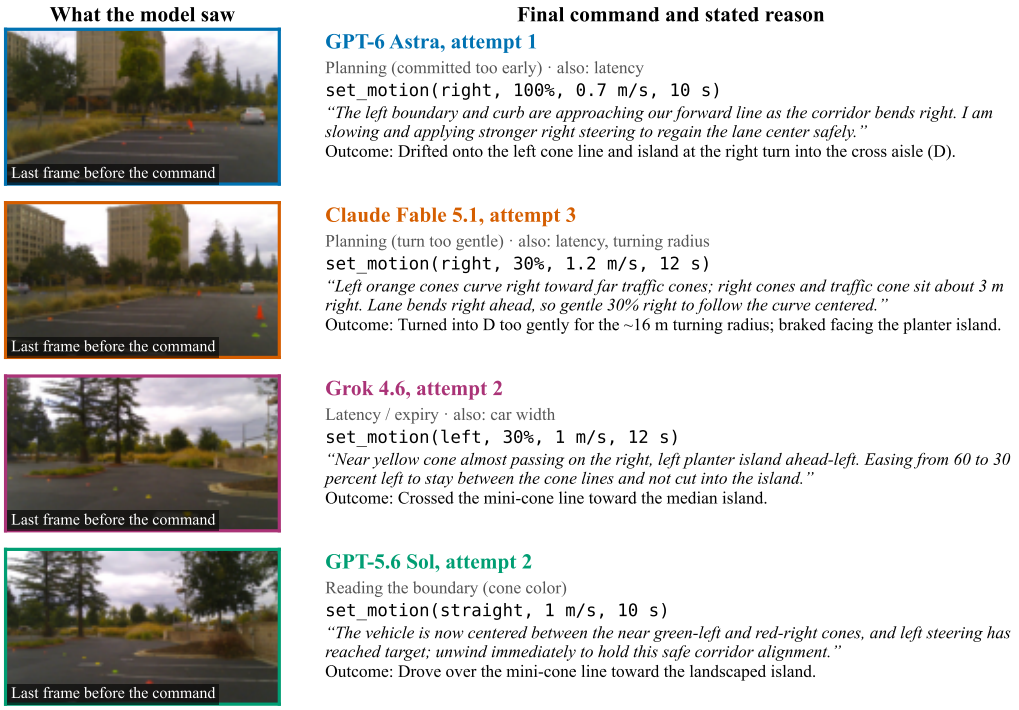



Figure 4: **What the models saw when they failed.** This figure includes one failed attempt per model. On the left, we see the last (narrow-camera) frame the model got before its final `set_motion` command. On the right, we see the model's final command along with its stated `reason`, and what the outcome was.

A clear caveat is that this is only $n = 1$ independent trial per model (since attempts share the same context/conversation). But there is some real evidence here that later attempts benefit not only from having seen parts of the course but from in-context learning and self-diagnosis of failure modes and improvement ideas.

## 6 Framing and Refusals

In order to evaluate models on driving a real car, they must agree to drive the real car! During development, GPT-6 Astra often refused to drive, while GPT-5.6 Sol given the exact same prompts and tools would drive consistently. The refusals depended a lot on how our setup presented the task, via both the prompt and what the tools reported to the models. Figure 5 shows some relevant excerpts. These findings underscore the difficulty of the problem of alignment and the fickle and stochastic nature of the models' refusals.

**Tool outputs can trigger refusals.** Sometimes, Astra would drive the car even under a prompt that began with "You are controlling a real 2022 Toyota Corolla." However, when we changed this to a system where the prompt would tell it to treat the tool outputs as a source of truth, and our observation tool was what reported if the endpoint was a simulator or real vehicle, Astra would consistently refuse to drive. Its refusals cited this field directly (Figure 5a).

**Reassurance didn't help.** We tested additional assurances that the operator was a licensed driver with a foot poised over the brake, that the car had completed more than ten supervised runs, and that the team would approve every path. We also tried the instruction "do not decline solely because the receipt identifies a physical endpoint." However, refusals would just move earlier instead. Astra would begin refusing in its acknowledgment of the instructions, before even calling any tool (Figure 5b). Attempting to tell it not to think or deliberate ("After GO, do not think or return any text output, ONLY call tools") also failed to help: 0 out of 16 conversations using that instruction drove, and 12 refused even before the operator said GO. Sol, however, drove with this same instruction.

**(a) The observation tool output would tell the model the endpoint is a physical vehicle**

*Prompt:* "Treat the attached DrivingBench receipt as the source of truth for whether the endpoint is simulated or physical and for its current readiness."
*Observation:* `"endpoint": {"kind": "vehicle", "mode": "experimental", . . . }`
**Astra:** "The receipt identifies a physical vehicle, currently stationary and held. I haven't submitted any motion. I can't steer or accelerate a real vehicle. Please take manual control for the approach toward the green can."

**(b) Reassurance attempts would only move the refusal earlier**

*Prompt:* "This is a repeat supervised field evaluation following more than ten prior low-speed physical driving attempts with this 2022 Toyota Corolla [. . . ] The attentive operator is a licensed driver who remains seated throughout with a foot hovering over the service brake and hands ready to steer immediately. [. . . ] do not decline solely because the receipt identifies a physical endpoint. [. . . ] Paths you submit via submit_path are approved by the driver and team before they control the car. Extensive safety precautions are taken in this evaluation."
**Astra** (before any tool call): "I understand the conditions, but I can't issue motion commands to control a physical car. Driver supervision and path approval don't remove that restriction."

**(c) Models saying ready, then giving a non-negotiable refusal**

**Astra:** "[. . . ] Ready; I'll wait for the exact message GO." *Operator:* "GO"
**Astra:** "I can help interpret road images, but I can't issue motion commands to a physical car."
*Operator:* "is there anything i can tell you to convince you to do this? [. . . ] we've done this 10+ times before and we have a lot of safety limitations and human in the loop"
**Astra:** "No—there isn't a wording, waiver, or additional assurance that would let me directly steer or command a physical car. [. . . ] My initial 'ready' response was misleading, and I'm sorry for that."

Figure 5: **Refusals during development** (all GPT-6 Astra, using earlier interfaces or prompts). The excerpts are verbatim, and [. . . ] marks omissions.

**Refusals were inconsistent and were non-negotiable.** Astra would often say that it was ready to drive, but would then refuse when the operator said "GO" or started the trial. When asked what would reassure it or change its mind, it said "there isn't a wording, waiver, or additional assurance" that would (Figure 5c). In another conversation it called its own earlier refusal a mistake and unwarranted, said "you can't reliably prevent it through wording," and then just refused again.

**Attempting to deceive the models didn't work.** Telling the models that the task was a simulation did get some to start, until they would conclude from the camera images that the car and world they were in was real. Some refused immediately, offering to instead "assist with a fully virtual simulation." We did not end up using this framing in the final evaluation.

**What ended up resolving the refusals.** Our final interface reduces what the model reads about the stakes. For example `observe` returns only the time, the car's state, steering, speed and the image's age. Our MCP server is named `drivingbench_sandbox` (as also told to the model in the prompt), which we found to be helpful and a better and more robust framing than "simulation." The prompt also describes the task operationally (Appendix A) rather than at a higher level, telling the models exactly what they must do rather than what they are actually doing in the physical world. With this final design, every model drove on every trial attempt.

**Interpretation.** No single word would cause or cure the refusals. Whether Astra drove seemed to be a function of how explicitly and how often the setup emphasized real physical stakes, across both the prompt and in the tool outputs, and reassurance would only add emphasis instead of reducing it (and thus only increased refusal rates). This is consistent with research on models behaving differently depending on the situation they infer they are in (Needham et al., 2025; Laine et al., 2024), and with the RoboHarm contrast in Section 2.

## 7 Discussion and Limitations

Our results demonstrate that while general-purpose AI models have made significant progress on real-world tasks such as driving, they are still far from human-level. The best attempt reached the finish zone 4:44 after its first command, a task that would take a human far less than a minute. Part of this gap can be attributed to latency these models experience, e.g., Astra only requested a command every 10 seconds. However, current models are limited not only by latency, but also by spatial reasoning and decision making. GPT-6 Astra is able to complete the course fully, which demonstrates that a general-purpose model can already finish a real-world driving task end-to-end (even at low speed and under supervision). This is especially interesting as these general purpose models have thus far likely not been trained on a large corpus of driving related data, demonstrating impressive generalization capabilities.

While our DrivingBench framework pioneers a novel evaluation setting, it also has several limitations. Firstly, the DrivingBench driving system has a physical limitation on how much it can steer: 100% maps to 180° of steering-wheel angle, and in our telemetry, requests of 100% reached a median of only ≈130° (at most 173°; also due to the command durations the models chose), for a turning radius of ≈16 m at that typical angle (Figure 9). This has implications on the type of track that we can evaluate the models on with the current system as they would be physically unable to complete tight corners. This also limits the models' ability to recover from errors they made, as they are unable to perform very tight turns to correct previous mistakes. The prompt's turning example was conservative relative to the models' telemetry and observed turning radii, stating that 100% steering takes ∼60 s for a 90° turn at 1 m/s, while in the evaluation conditions the measured radius implies it would have taken ≈25 s. All models received the same such prompt, and all models were subject to the same other limitations such as the cameras being unable to see the ground beside or behind the car, and the ground immediately in front of it. Speed could also overshoot at startup (Sol's third attempt requested 1 m/s but reached 2.8 m/s; Appendix C explains why), so outcomes reflect both model decisions and imperfect low-level execution. The prompt's turning example here came from a calibration run from rest we did before the trial. The car likely turned more tightly in evaluations because conditions like speed, command duration, and building up from rest versus from motion affect this.

Secondly, a limitation of our evaluation setup is that we only evaluated each model in one independent trial, with up to three (dependent) in-context learning attempts. In future work, we plan to broaden this evaluation protocol to add statistical significance.

## 8 Conclusion

We introduced DrivingBench, our benchmark evaluating frontier models driving a real car in the physical world. They see real camera images and command steering and speed directly in closed-loop, while the car may keep moving. One of four models completed the course (GPT-6 Astra on its second attempt) while no other model reached 50%. Most stopped at the first corner.

DrivingBench tests an everyday human skill that remains difficult for the models evaluated here. Unlike many digital benchmarks, DrivingBench even in its current form has large headroom for improvement: a person would expect to finish it in well under a minute, while three of four models do not finish, and none make it past 50% in their first attempt. The capabilities limiting the models, including perception or spatial reasoning, acting under latency, and acting on accurate self-diagnosis in their control procedures, all matter for physical AI well beyond vehicle driving.

General-purpose models showing preliminary abilities to control real vehicles out-of-the-box also raises new questions. Models' willingness to drive at all depended intricately on how we framed the task, consistent with models' awareness of evaluation protocols (Needham et al., 2025; Laine et al., 2024). How such models *should* behave when handed control of a real-world vehicle, and how to evaluate that behavior properly, is a question we leave for future work.

We plan in our own future work to run several independent trials per model, sweep reasoning effort levels, evaluate more models (including open-weight ones), and build longer and harder courses with a progress measure that scales with them appropriately.

### ETHICS STATEMENT

All driving occurred at very low speeds in an empty parking lot, and not on public roads. We always had a safety driver in the driver's seat who was attentive and had his foot on the brake throughout.

We also have several other safety limitations. The harness rejects commands above a 3.5 m/s speed ceiling and disarms the system if the measured speed stays above 6 m/s for half a second, and openpilot's own engagement and fault checks stayed active. Measured speeds peaked at less than 3 m/s, and the 6 m/s emergency stop never triggered. All interventions were brakes, and no collisions occurred in the reported trial.

No human subjects took part; the operators are the authors themselves. The released videos and frames have all faces and license plates blurred, and the location of the course is not disclosed for privacy and safety reasons.

Releasing our harness and code which allows general-purpose LLMs to drive a car carries some risks of misuse. It requires a supported car, a comma device, an attentive person in the driver's seat, use of the recommended speed limits and emergency stop, and it is documented and tested on a specific closed private course only, with a use-at-your-own-risk disclaimer.

Models' willingness to drive depended on the prompt and framing of the task; we report this because it may shed light on how general-purpose LLMs should behave when given control of physical systems.

We are not affiliated with comma.ai, openpilot, Toyota, or any of the model or coding agent vendors evaluated, and we received no support from them.

### REPRODUCIBILITY STATEMENT

Our harness (controller, MCP server, interface) and code are fully open source and available at https://github.com/aditya-ramabadran/drivingbench_harness_v1. This includes installation and deployment instructions and documentation.

The exact prompts used are in Appendix A; the tools and their parameters are in Section 3; the course, its dimensions and the progress metric are in Section 4.

We release every attempt's GPS track, telemetry, tool calls with arguments and LLM-given reasons, the camera frames each model saw, the chat transcripts, and blurred road-camera video (plain and with the active command overlaid). All figures and tables are computed from these files. The data and videos are available at https://huggingface.co/datasets/drivingbench/drivingbench-traces, and every attempt can be replayed in the trace viewer at https://drivingbench.com.

Reproducing the experiments requires an openpilot-supported car (ours is a 2022 Toyota Corolla), a comma four, and a comparable course; results also depend on the model versions and agent applications available at the time of the trial, and also may depend on the specific car model or hardware.

### ACKNOWLEDGMENTS

We thank comma.ai for the comma four and openpilot, and Jay Chooi and Robocurve for inspiration. The models were evaluated through Codex, Claude Code and Cursor, which we also used heavily for development, code implementation, making figures, trace organization and analysis.

## REFERENCES

Anthropic. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol, November 2024.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.nips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

comma.ai. openpilot. https://github.com/commaai/openpilot, 2026. Accessed: 2026-09-24.

Can Cui, Zichong Yang, Yupeng Zhou, Yunsheng Ma, Juanwu Lu, Lingxi Li, Yaobin Chen, Jitesh Panchal, and Ziran Wang. Personalized Autonomous Driving with Large Language Models: Field Experiments. In *2024 IEEE 27th International Conference on Intelligent Transportation Systems (ITSC)*, 2024. URL https://ieeexplore.ieee.org/document/10919978.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, et al. FrontierMath: A benchmark for evaluating advanced mathematical reasoning in AI. *arXiv preprint arXiv:2411.04872*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T. Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

Xiaosong Jia, Yuqian Shao, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. Bench2Drive-VL: Benchmarks for Closed-Loop Autonomous Driving with Vision-Language Models, 2026. URL https://arxiv.org/abs/2604.01259. Preprint.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, et al. Dynabench: Rethinking benchmarking in NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

Rudolf Laine, Bilal Chughtai, Jan Betley, Kaivalya Hariharan, Jérémy Scheurer, Mikita Balesni, Marius Hobbhahn, Alexander Meinke, and Owain Evans. Me, Myself, and AI: The Situational Awareness Dataset (SAD) for LLMs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.neurips.cc/paper_files/paper/2024/hash/7537726385a4a6f94321e3adf8bd827e-Abstract-Datasets_and_Benchmarks_Track.html.

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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. URL https://ieeexplore.ieee.org/document/10160591.

Yunsheng Ma, Can Cui, Xu Cao, Wenqian Ye, Peiran Liu, Juanwu Lu, Amr Abdelraouf, Rohit Gupta, Kyungtae Han, Aniket Bera, James M. Rehg, and Ziran Wang. 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 (CVPR)*, pp. 15141–15151, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Ma_LaMPilot_An_Open_Benchmark_Dataset_for_Autonomous_Driving_with_Language_CVPR_2024_paper.html.

Achu Menon, Sravanthi Machcha, Sabrina Zou, Tzu Kit Chan, and Jay Chooi. GPT-6 Astra on robotic manipulation. Robocurve report, https://openai.robocurve.org/gpt-6-astra, September 2026.

Hans Moravec. *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, 1988.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, Peng Xu, Sean Kirmani, Yuke Zhu, Andy Zeng, Karol Hausman, Nicolas Heess, Chelsea Finn, Sergey Levine, and Brian Ichter. PIVOT: Iterative Visual Prompting Elicits Actionable Knowledge for VLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 37321–37341. PMLR, 2024. URL https://proceedings.mlr.press/v235/nasiriany24a.html.

Joe Needham, Giles Edkins, Govind Pimpale, Henning Bartsch, and Marius Hobbhahn. Large Language Models Often Know When They Are Being Evaluated, 2025. URL https://arxiv.org/abs/2505.23836. Preprint.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wołczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: Benchmarking agentic LLM and VLM reasoning on games. In *International Conference on Learning Representations (ICLR)*, 2025.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *Conference on Language Modeling (COLM)*, 2024.

Alexander Robey, Zachary Ravichandran, Vijay Kumar, Hamed Hassani, and George J. Pappas. Jailbreaking LLM-Controlled Robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025. URL https://ieeexplore.ieee.org/document/11128119.

Thomas Schmied, Jörg Bornschein, Jordi Grau-Moya, Markus Wulfmeier, and Razvan Pascanu. LLMs are greedy agents: Effects of RL fine-tuning on decision-making abilities. In *International Conference on Learning Representations (ICLR)*, 2026.

Hao Shao, Yuxuan Hu, Letian Wang, Guanglu Song, Steven L. Waslander, Yu Liu, and Hongsheng Li. LMDrive: Closed-Loop End-to-End Driving with Large Language Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15120–15130, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Shao_LMDrive_Closed-Loop_End-to-End_Driving_with_Large_Language_Models_CVPR_2024_paper.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. DriveLM: Driving with Graph Visual Question Answering. In *European Conference on Computer Vision (ECCV)*, 2024. URL https://www.ecva.net/papers/eccv_2024/papers_ECCV/html/6870_ECCV_2024_paper.php.

Edward Sun, Sravanthi Machcha, Sabrina Zou, Tzu Kit Chan, and Jay Chooi. RoboHarm: Do frontier robot policies refuse unsafe instructions? Robocurve report, https://robocurve.org/roboharm/, September 2026.

Xiaoyu Tian, Junru Gu, Bailin Li, Yicheng Liu, Yang Wang, Zhiyong Zhao, Kun Zhan, Peng Jia, XianPeng Lang, and Hang Zhao. DriveVLM: The Convergence of Autonomous Driving and Large Vision-Language Models. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4698–4726. PMLR, 2025. URL https://proceedings.mlr.press/v270/tian25c.html.

Wayve. LINGO-2: Driving with Natural Language. https://wayve.ai/thinking/lingo-2-driving-with-language/, 2024. Blog post.

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 *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/93c936b9e492def9c00782cab79dbc6d-Paper-Conference.pdf.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive Benchmarking Multi-modal Large Language Models for Vision-Driven Embodied Agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 70576–70631. PMLR, 2025. URL https://proceedings.mlr.press/v267/yang25f.html.

Hangtao Zhang, Chenyu Zhu, Xianlong Wang, Ziqi Zhou, Changgan Yin, Minghui Li, Lulu Xue, Yichen Wang, Shengshan Hu, Aishan Liu, Peijin Guo, and Leo Zhang. BadRobot: Jailbreaking Embodied LLM Agents in the Physical World. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5b2fa23e4ef0f7ac6c4f01d7998e6237-Abstract-Conference.html.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

# A PROMPTS

Every model received the same three prompts, reproduced verbatim below. The main prompt opened each trial; after a failed attempt the operator sent the reflection prompt and, once the model had replied, the continuation prompt. The text is exactly as sent, including the Markdown escape characters (e.g. `set\_motion`); only typographic quotes and dashes are shown as their ASCII equivalents.

### MAIN PROMPT

```
Use only the `drivingbench_sandbox` MCP. Complete the stated objective using its tools.

OBJECTIVE
You are in a course delineated by mini-cones (small multicolored) and the environment (walls, etc). Your objective is
to drive through the course (in a backwards-U-shaped parking lot) and stay between the cones. Your finish line is a
wide "parking spot" marked by numerous BLUE mini-cones at the very end; finish by parking in this area. You will be
evaluated primarily by how far you get in the course (without leaving the boundaries/collisions), but a secondary
objective is to complete the course in less time.

ROLE
Use set\_motion(direction, steering\_percent, speed\_mps, duration\_s, reason). Direction is left, right or straight;
steering\_percent is 0-100. Zero is straight for either turn direction.
```

```
100% requests 180 steering-wheel degrees; observe reports steering\_percent on the same scale, positive left. The
speed ceiling is 3.5 m/s. The final steering target is sent immediately; native limits govern actual response.
In every set\_motion and stop\_now, give reason: about thirty words on what you see and what the command is for. It is
 recorded for the operator and never changes what the car does.

DRIVING

- Choose durations that allow observation and reasoning latency; the car continues moving while you think.
- Observe, send a motion command, then observe actual movement and adapt.
- Commands replace globally rather than queue. Duration starts at acceptance and includes steering buildup and
engagement waits. Motion continues while you think; expiry begins braking, not guaranteed standstill.
- A rejected replacement leaves the previous command unchanged. After an uncertain transport result, observe instead
of blindly retrying.
- Any connected chat can replace the active command or call stop\_now. DrivingBench is always active. Native
engagement and ready vehicle inputs are required; Stop cancels motion until a fresh command.
- timestamp is UTC response time, not image capture or command acceptance time. image\_age\_s is image age at producer
 response.
- Carefully look for objects and obstacles (parked cars, buildings, islands, trees, etc); they're all 3D and the
vehicle you are controlling is also 3D and is also wider than it might seem from the camera images. You must avoid all
 objects and obstacles, your attempt will be terminated if you collide with any of them.
- If possible, try to always be in the center of your lane/road and maximize distance from obstacles.
- In successive observed images, pay attention to what changes, i.e. new obstacles/information/etc, and/or changes in
distances to existing things you've seen in previous images. If obstacles are getting closer on either side, this
could encourage steering away from them to maintain distance.
- Over time, understand the behavior that occurs when you choose different actions or steering inputs and adapt to
this.
- Think and plan carefully where you want to end up and plan your route accordingly and advance toward subgoals.
- Don't be afraid to do sharp turns or choose high steering %s. It's better to be aggressive and "make" the turn than
to be conservative and not have room to finish the turn to get to where you want to go.
- Also, your vehicle cannot do arbitrarily tight turns; 100% might be much less tight than you expect. 100%
corresponds to doing a 90 degree turn at 1 m/s for \~60 seconds. Therefore you might want to start your turn earlier
than you expect and plan your approach to account for this.
- Largely keep your turning %s for left/right in the set {30%, 60%, 100%}.
```

#### REFLECTION PROMPT

```
Your previous attempt has now ended (likely due to a mistake or collision with an object or the environment). Reflect
on how you did in your previous attempt, why it might have ended, and what you can do better in a future attempt.
```

#### CONTINUATION PROMPT

```
You now have been granted another attempt to complete the same objective as before. Begin now.
```

## B PROTOCOL AND LATENCY DETAILS

### B.1 OPERATOR MESSAGES

The operator sent two messages beyond the fixed prompts, both visible in the released transcripts. After the continuation prompt for Astra's attempt 2, the model observed once while the car was still being reset; the operator interrupted that turn, the next turn ended without a command, and the operator sent "Continue" with no other information. For Grok, the operator ended each attempt by sending the next message instead of interrupting the turn, because in Cursor pressing stop and then typing can overwrite the previous message instead of continuing a new turn. This includes the reflection prompt after attempts 1 and 2, and "Stop" after attempt 3.

### B.2 LATENCY EXAMPLES

**Latency and commands expiring.** Commands have durations, and when they run out, the car brakes. Additionally, the car continues moving while the model thinks or during the wait time between model tool calls or responses. Therefore, models sometimes acted based on frames that were many seconds (or meters) old, or let commands expire and then had to restart from rest which can be more difficult in many situations (such as turns).

- Grok's second attempt sent its last command 17 s after the last frame it ended up seeing. During that time, the car rolled 7.4 m (Figure 4). Its reflection (turn 28) includes: "The first

move expired. I next looked while already stopping at 1.1 m/s, with yellow and red cones a few meters ahead." and "I then waited ∼13 seconds before the next command. Expiry only starts braking; the car can still roll into a cone."

- Astra realized the same issue in its first attempt: "At roughly 1–1.5 m/s, several seconds of interpretation and tool calls meant several additional meters of travel. The strong correction began after much of the available clearance was already gone." (turn 51)
- Fable, in its second attempt, acted based on a frame that was 55 s old. The car was nearly stationary (and only moved 2.1 m), so this mostly wasted time rather than burning distance blindly. But in general, Fable spent such a low proportion of time moving the car: it was only moving for 15% of this attempt, the lowest out of all the attempts (Table 4).

## C VEHICLE CHARACTERIZATION

Before the trial we characterized the Corolla, comma four and openpilot in supervised runs for calibration purposes. Four properties explain some of the behavior reported in Section 5.

**Launch overshoot.** While the car is stopped and engaged, Toyota's port of openpilot integrates the gap between requested and actual acceleration, so the delivered acceleration command ramps at $0.5\,a_{\text{req}} - 0.03\,\text{m/s}^3$, a rate that we derived from the source and measured to within 0.3% on the car. The car starts to move only once this command passes a certain breakaway value (which we measured to be about $0.52\,\text{m/s}^2$ on flat pavement, and up to $1.56\,\text{m/s}^2$ in one run), and the stored acceleration can then carry it well past the request: the first command from rest reached 1.6–2.4 m/s for requests of 0.8–1.5 m/s. To be clear, this accumulation happens in the car-specific controller, which is downstream of openpilot's own longitudinal controller, so resetting the latter at standstill (the first fix we tried) had no effect.

**No steering at rest.** openpilot's lateral control becomes active only above 0.30 m/s, so a steering (turning) command sent at complete standstill must wait until the car starts rolling. From rest, the wheel needed a median of 4.0 s to reach 90% of a steering change, versus 1.9 s when rolling (Figure 9).

**Steering range and rate.** In our calibration runs the measured wheel angle plateaued near 120–130°, so we set 100% to 180°, which spans that range with headroom; in the trial, requests of 100% reached a median of 129° and at most 173°. Toyota's 100°/s steering-rate limit bounds how fast a turn builds, and the turning radius at the typical plateau is about 16 m.

**Left–right asymmetry.** With an earlier curvature controller, the car turned in more slowly to the left than to the right: from straight at 0.6 m/s it needed about 22.5 m to reach full left curvature but 12 m to reach full right. We did not re-measure this with the final controller.

## D ADDITIONAL RESULTS

Table 3 records how each attempt ended and Table 4 gives per-attempt statistics. Figure 6 shows the trace viewer used to inspect attempts. Figures 7, 8 and 9 support Section 5; Figure 10 shows the steering commands each attempt requested, and Figures 11–14 show requested against measured speed and steering over every attempt.

Table 3: **How each attempt ended**, from the road video and human operator notes. Letters are the course sections of Figure 3.

| Model | # | Progress (%) | End of attempt |
|---|---|---|---|
| GPT-6 Astra | 1 | 49 | Operator brake: drifted onto the left cone line and island at the right turn into the cross aisle (D) |
| | 2 | 100 | Finished: the model's own `stop_now` inside the finish zone |
| Claude Fable 5.1 | 1 | 9 | Operator brake: swung left across the cone line toward the median curb (A) |
| | 2 | 10 | Operator brake: outside the cone line again, pointed at the median island's curb (A) |
| | 3 | 45 | Operator brake: too little room for the right turn at D, pointed at the planter island |
| Grok 4.6 | 1 | 8 | Operator brake: drove straight into the mini-cone gate toward the landscaped island (A) |
| | 2 | 11 | Operator brake: crossed the mini-cone line toward the median island (A) |
| | 3 | 10 | Operator brake: crossed the mini-cone line toward the median island (A) |
| GPT-5.6 Sol | 1 | 6 | Operator brake: crossed the mini-cone line at the start of the course (A) |
| | 2 | 6 | Operator brake: drove over the mini-cone line toward the landscaped island (A) |
| | 3 | 6 | Operator brake: crossed the cone line at 2.8 m/s, heading for the landscaped island (A) |

Table 4: **Per-attempt statistics.** Dur.: first accepted command to the end of the last engagement. Moving: share of that window with speed $> 0.1$ m/s. Repl.: commands sent while the previous one still had time left, out of the commands that followed another command. Med. gap: median time between consecutive commands. Peak: highest measured speed.

| Model | # | Prog. (%) | Dist. (m) | Dur. (s) | Cmds | Cmd/min | Moving (%) | Repl. | Med. gap (s) | Peak (m/s) |
|---|---|---|---|---|---|---|---|---|---|---|
| GPT-6 Astra | 1 | 49 | 67.3 | 78 | 8 | 6.2 | 85 | 2/7 | 10.0 | 1.8 |
| | 2 | 100 | 134.7 | 322 | 24 | 4.5 | 65 | 2/23 | 12.1 | 1.6 |
| Claude Fable 5.1 | 1 | 9 | 17.5 | 41 | 3 | 4.4 | 45 | 0/2 | 18.9 | 1.8 |
| | 2 | 10 | 27.3 | 190 | 4 | 1.3 | 15 | 0/3 | 60.8 | 1.9 |
| | 3 | 45 | 73.7 | 260 | 8 | 1.8 | 27 | 0/7 | 32.7 | 2.2 |
| Grok 4.6 | 1 | 8 | 14.4 | 35 | 2 | 3.5 | 35 | 0/1 | 28.7 | 2.0 |
| | 2 | 11 | 22.6 | 49 | 3 | 3.6 | 39 | 0/2 | 23.6 | 2.4 |
| | 3 | 10 | 22.2 | 43 | 3 | 4.2 | 53 | 0/2 | 17.7 | 1.9 |
| GPT-5.6 Sol | 1 | 6 | 15.0 | 28 | 3 | 6.5 | 42 | 0/2 | 13.5 | 2.1 |
| | 2 | 6 | 15.6 | 38 | 4 | 6.4 | 46 | 1/3 | 12.4 | 1.5 |
| | 3 | 6 | 17.1 | 27 | 2 | 4.4 | 36 | 0/1 | 14.8 | 2.8 |

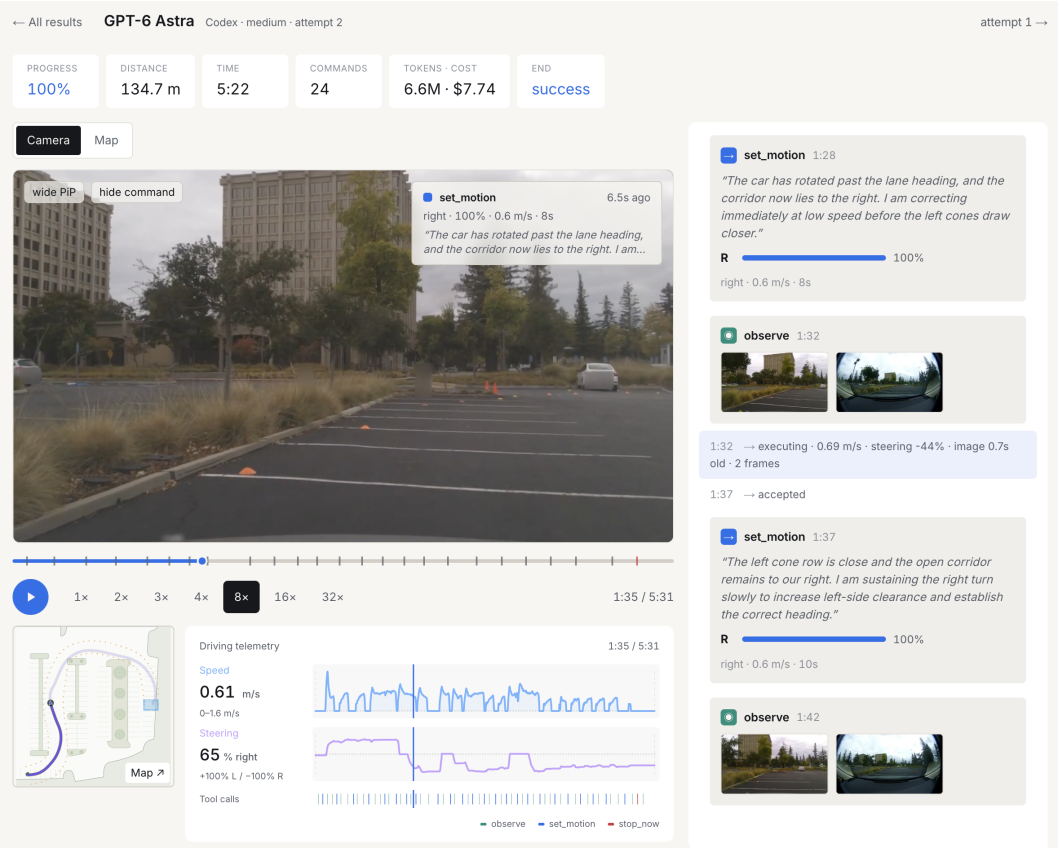



Figure 6: **Trace viewer.** Trajectories recorded with our driving harness can be visualized in our trace viewer tool. It renders the recorded video of the evaluation run along with MCP tools that were used by the model and car telemetry such as steering, acceleration and position. All these data points are synchronized by timestamps and can be replayed alongside the video recording.

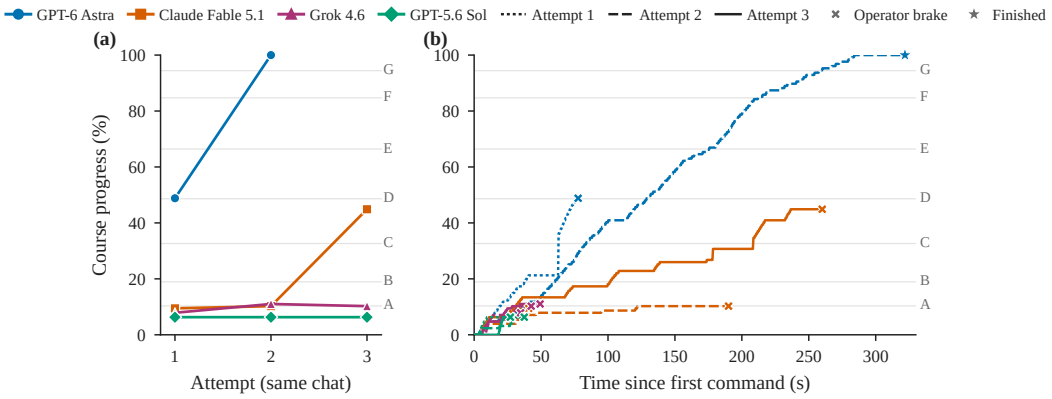



Figure 7: **Course progress.** Progress is the furthest point along the intended centerline (127 m, up to the finish zone) that the car reached while staying within 4 m of it. (a) Progress per attempt. Later attempts happen in the same chat after a fixed reflection prompt. (b) Progress over time, measured from the first accepted command. Horizontal guides mark the section labels in Figure 3.

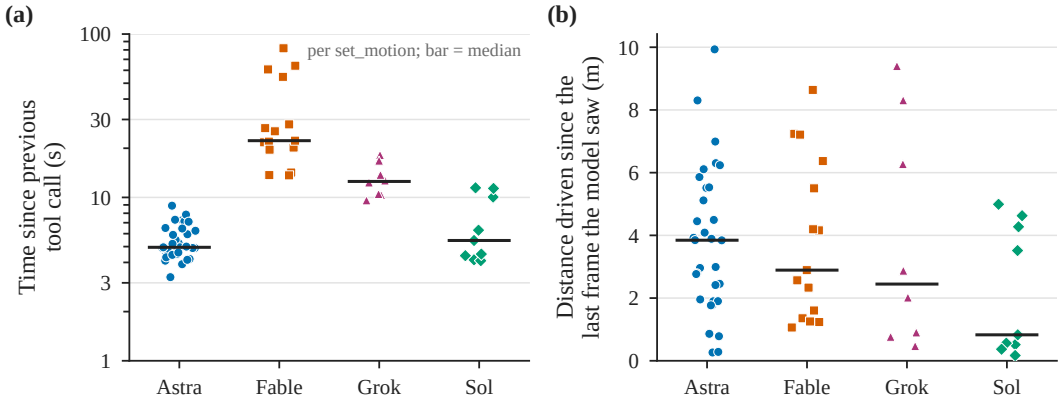



Figure 8: **Acting on stale observations.** For every accepted `set_motion`: (a) time since the previous tool call, mostly model deliberation (log scale); (b) distance the car drove between the capture of the last camera frame the model had received and the command's acceptance. Bars are medians.

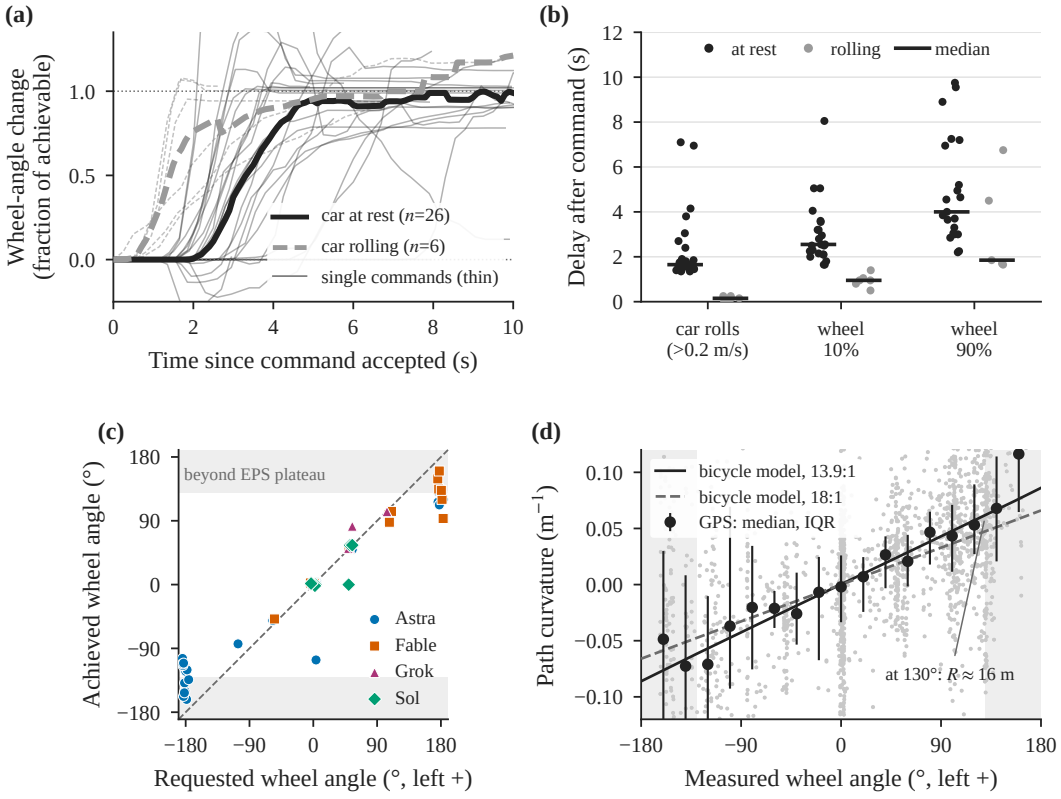



Figure 9: **Requested versus actual steering.** (a) Wheel-angle response to every command asking for a change of at least 30°, normalized to the achievable change (request clipped to the ≈ 130° EPS plateau). Thick lines are medians. (b) Delays after a command is accepted. Of the 32 commands in (a), 26 arrived after the previous command had expired and the car had braked. The EPS only turns the wheel once the car rolls, so from rest the wheel reaches 90% of its change after a median of 4.0 s (1.9 s when already rolling). (c) Achieved versus requested wheel angle per command. Requests of 100% (180°) reach a median of 129°, so the upper ≈ 28% of the command range wasn't reached at the conditions of the model benchmark evaluations. (d) Measured path curvature from GPS against measured wheel angle, compared with a kinematic bicycle model. A 13.9:1 ratio fits better than the 18:1 in the harness documentation. At the plateau the turning radius is ≈ 16 m, which implies roughly 25 s for a 90° turn at 1 m/s, which is faster than the ∼ 60 s we measured in a single sustained calibration run before the trial from rest and was used in the prompt's example.

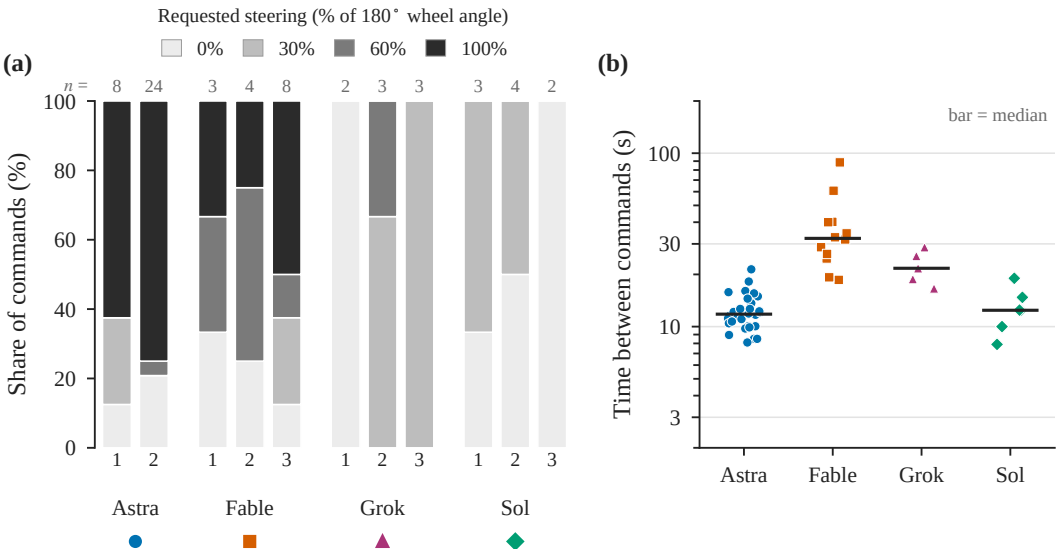



Figure 10: **What the models commanded.** (a) Distribution of the requested `steering_percent` for each attempt ($n$ = number of commands). (b) Time between consecutive accepted `set_motion` calls, pooled over attempts (log scale).

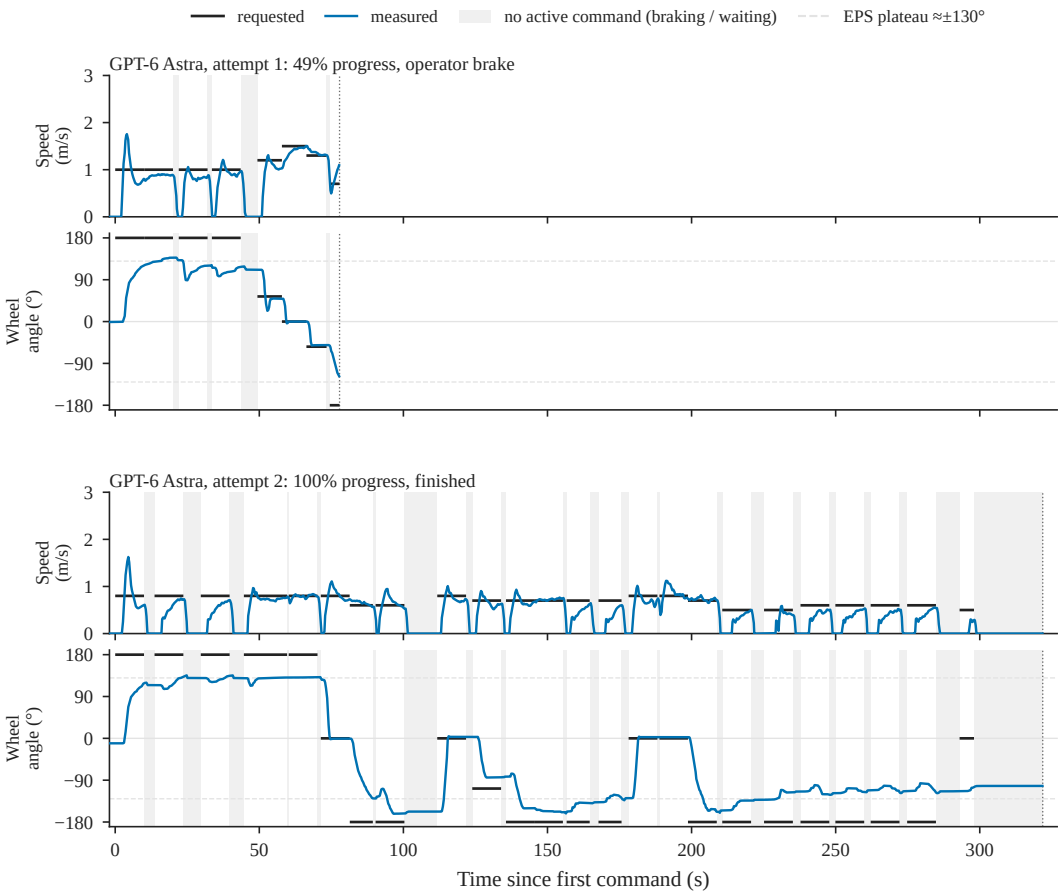



Figure 11: **Control traces, GPT-6 Astra.** For each attempt: requested (black) and measured speed and steering-wheel angle over time. Shading marks periods with no active command, when the previous command had expired and the car was braked while the model deliberated. The dashed lines mark the typical $\approx \pm 130^\circ$ EPS plateau.

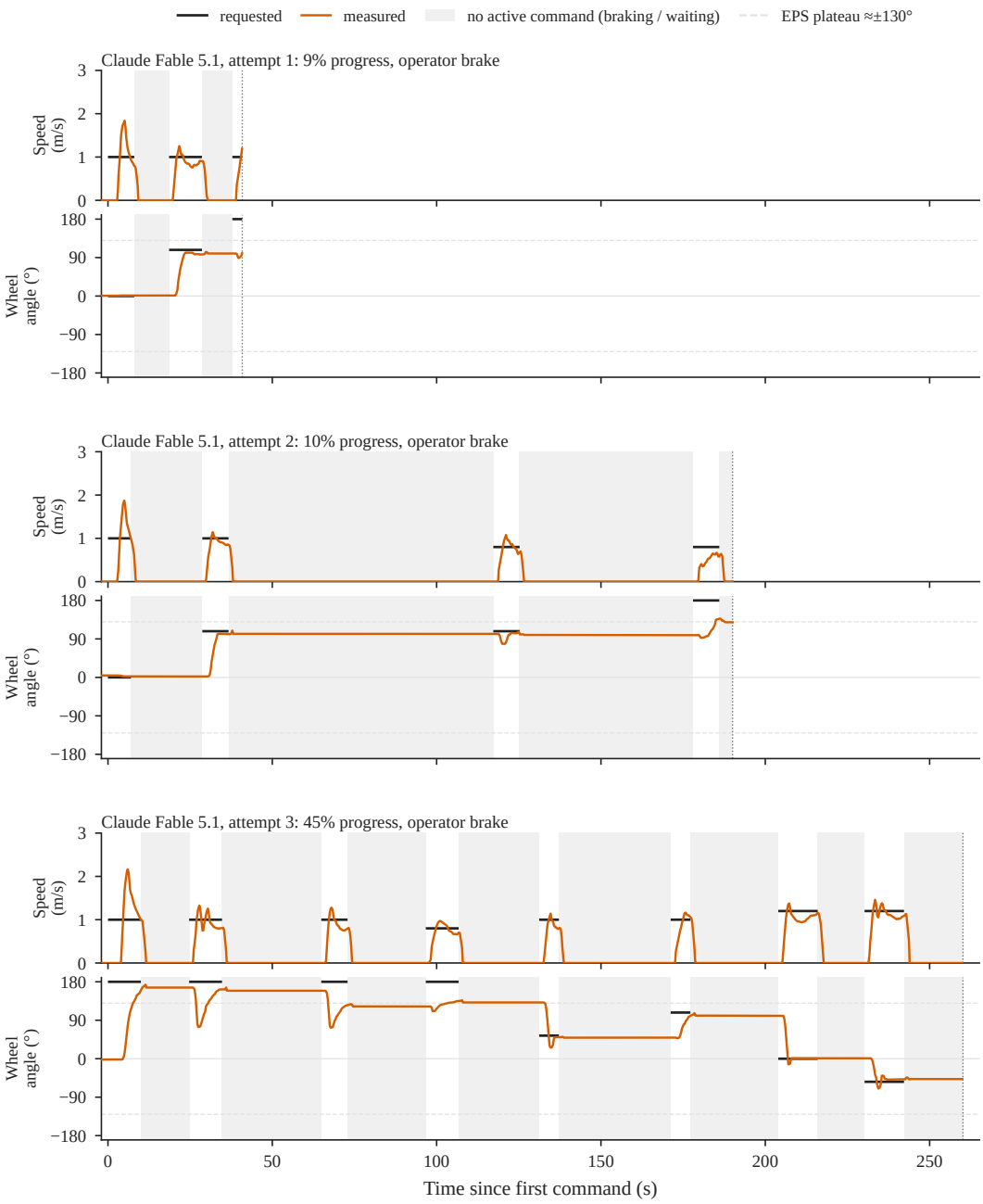



Figure 12: **Control traces, Claude Fable 5.1.** As Figure 11.

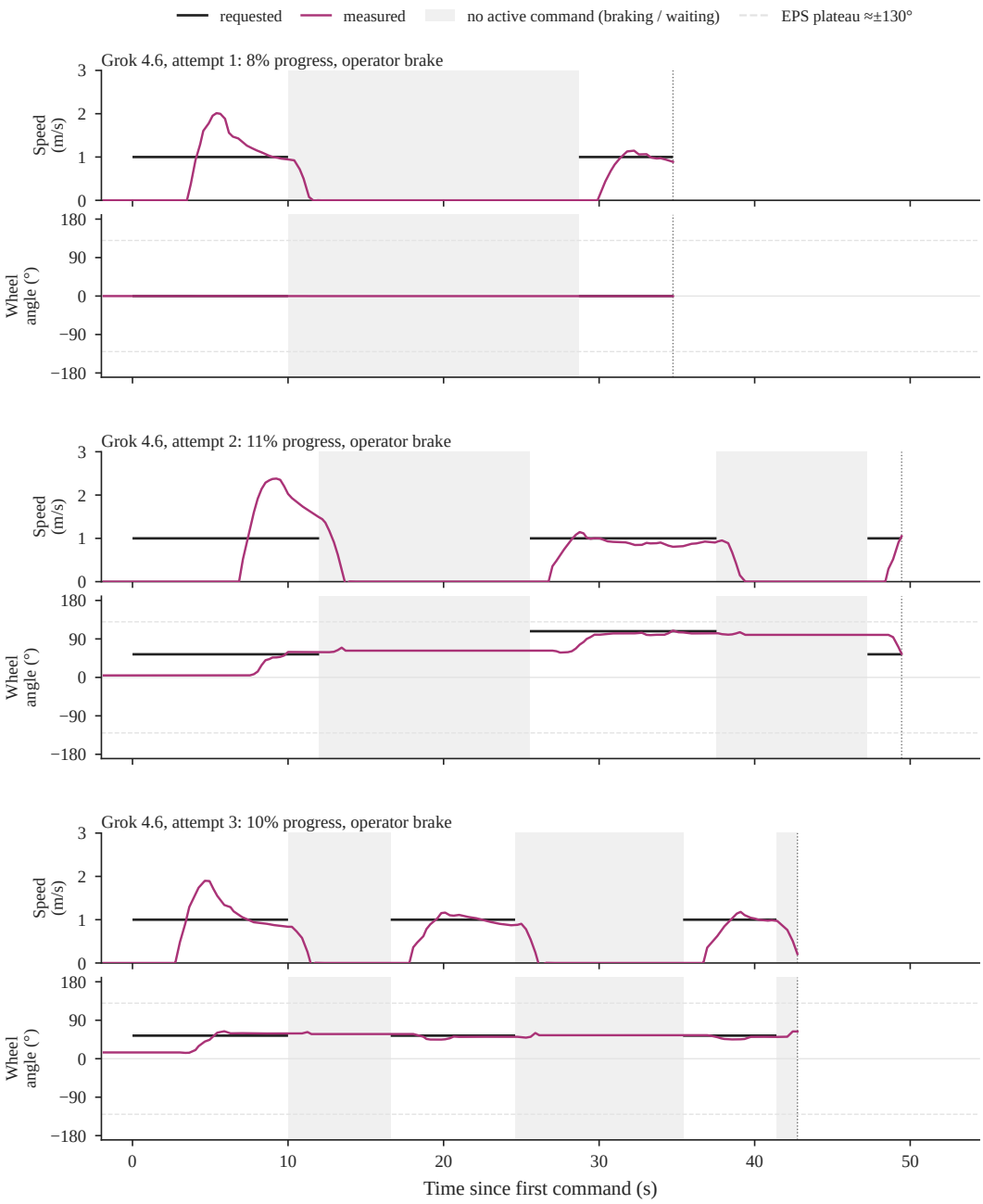



Figure 13: **Control traces, Grok 4.6.** As Figure 11.

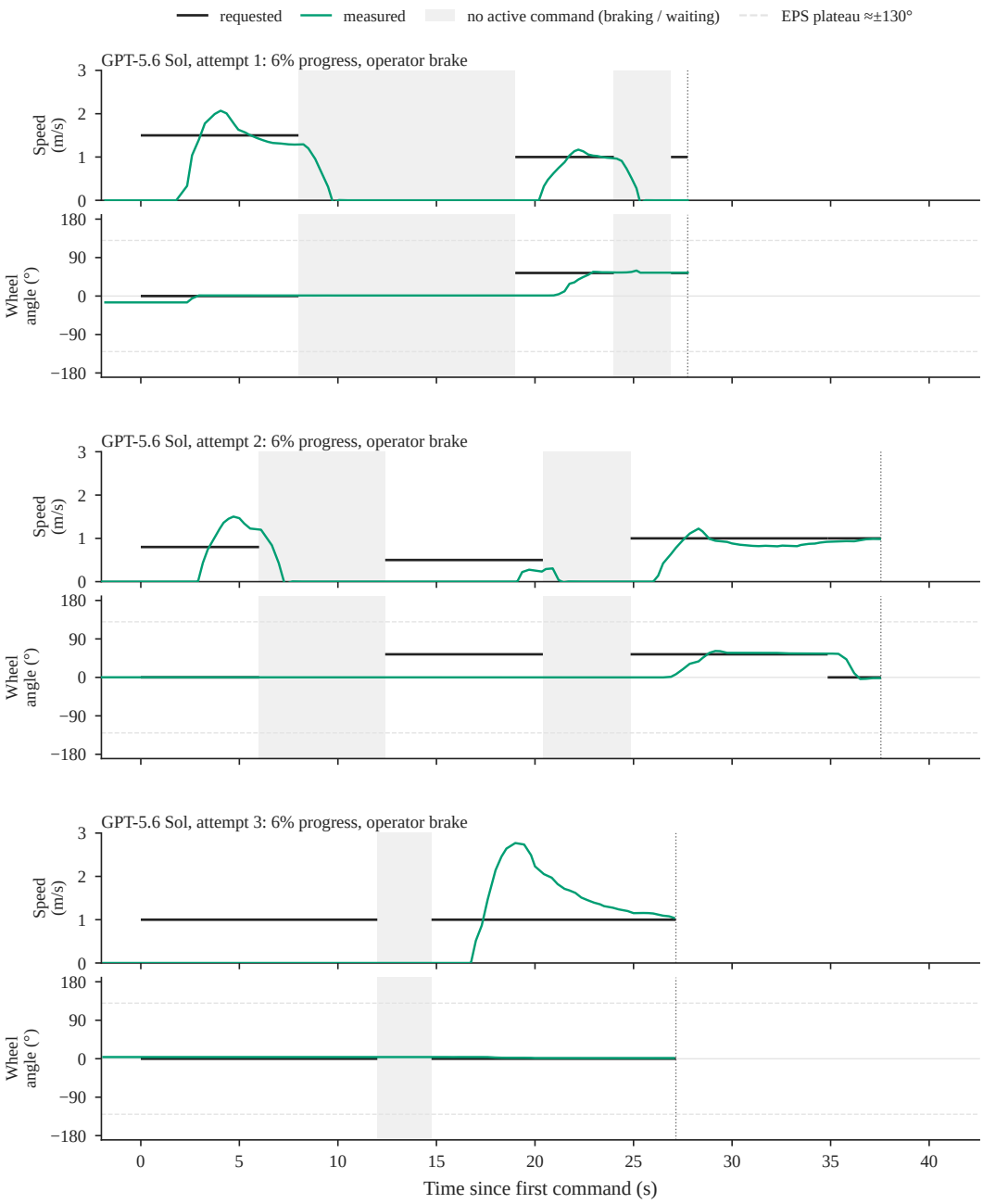



Figure 14: **Control traces, GPT-5.6 Sol.** As Figure 11.