GridLM: An LLM-Based Approach to Gameplay Performance Coaching in Forza Motorsport

Kyle Shervington
kyle.shervington@ucf.edu
University of Central Florida
Orlando, Florida, USA

Nafisa Islam
na942507@ucf.edu
University of Central Florida
Orlando, Florida, USA

ABSTRACT

GridLM introduces a novel integration of large language models (LLMs) with telemetry data to deliver personalized gameplay coaching for Forza Motorsport. This system bridges the skill gap between casual and competitive players by transforming complex in-game metrics, such as speed and throttle, into actionable performance insights. Leveraging a decoupled architecture, GridLM processes real-time telemetry data through a structured pipeline and MongoDB-based storage, subsequently generating tailored recommendations via LangChain-integrated LLMs. Evaluations demonstrate the system's effectiveness in providing coherent feedback, addressing token efficiency, and reducing hallucinations, establishing GridLM as a promising tool for advancing gameplay coaching analytics. Future expansions aim to enhance real-time analysis and multimodal data integration.

KEYWORDS

gameplay analytics, large language model, forza motorsport racing, personalized coachin, telemetry, langchain

1 INTRODUCTION

GridLM is an innovative approach to gameplay coaching which introduces an intelligent, data-driven system that improves how players approach skill improvement in gaming. Using Forza Motorsport as a case study, GridLM integrates cutting-edge Large Language Models (LLMs) with in-game, vehicle telemetry data to provide personalized feedback aimed at improving player performance. This system was designed to bridge the significant skill gap between casual and competitive players by offering actionable insights traditionally gained through extensive practice or expert analysis.

GridLM utilizes a robust data pipeline that ingests and processes telemetry data such as speed, throttle percentage, braking pressure, and lap segmentation, into a structured format suitable for LLM interpretation. Cloud-based, time series storage solutions are used to enable seamless data retrieval and analysis. By integrating LangChain, GridLM employs prompt engineering to deliver specific coaching suggestions, allowing players to refine their driving techniques and reduce lap times.

GridLM was optimized by addressing key challenges related to token efficiency, response latency, and hallucination reduction; the result is a system that remains cost-effective and reliable for frequent use. Initial evaluations revealed the system's ability to provide coherent and actionable recommendations, validating its potential as a transformative tool for gameplay improvement. Furthermore, by leveraging the Llama 3.2 family of models alongside advanced pre-processing methods, GridLM demonstrates the capacity to simplify complex telemetry patterns into meaningful insights, fostering an accessible and user-friendly experience.

As a completed proof-of-concept, GridLM sets a new standard for personalized gaming analytics, with future potential to expand into broader data categories such as grip levels and driving consistency. This project not only highlights the applicability of LLMs in gaming but also showcases their potential to create interactive, tailored learning experiences in the rapidly evolving world of competitive e-sports.

2 RELATED WORK

The integration of Large Language Models (LLMs) into gaming has opened new avenues for enhancing player experiences through intelligent, responsive systems. In the realm of racing simulations, Forza Motorsport has been at the forefront of incorporating advanced machine learning techniques to elevate realism and player engagement. The latest installment by Turn 10 Studios utilizes machine learning to develop sophisticated AI opponents, known as "Drivatars," which learn and adapt to player behaviors, creating a more dynamic and personalized racing environment. [9]

Beyond racing games, LLMs have been employed across various gaming genres to enrich interactions and gameplay. A comprehensive survey by Arxiv.org discusses the multifaceted roles of LLMs in games, including serving as in-game assistants, non-player characters (NPCs), and even game masters, thereby enhancing narrative depth and player immersion. [3]

In the context of game-playing agents, LLMs have demonstrated significant potential. Research has shown that LLMs can function as game players, NPCs, or assistants, providing hints and managing tasks within the game environment. This versatility underscores the capability of LLMs to understand and generate contextually relevant content, thereby enriching the gaming experience. [4]

The application of LLMs in gaming is not limited to enhancing player interaction but also extends to game development processes. For instance, AI-driven systems have been utilized to generate game assets, design levels, and create dynamic narratives, thereby streamlining development and introducing innovative gameplay elements. [5]

In summary, the convergence of LLMs and gaming, exemplified by initiatives like Forza Motorsport's AI integration and broader applications across various game genres, highlights a transformative shift in how games are developed and experienced. These advancements not only enhance realism and engagement but also pave the way for more personalized and immersive gaming experiences.

3 METHOD

GridLM is a decoupled system that ingests and processes data and later retrieves that data for additional processing and LLM interpretation. It is decoupled in the sense that the data collection infrastructure operates independently of the LLM-driven assessment system. Data collection and storage must be performed and completed before initiating LLM operations. This section will detail the creation and operation of each component of the GridLM system, starting from telemetry data generation and collection to the final output of LLM-created coaching reports.

3.1 System Overview

See Figure 6 in Appendix A for the full architecture diagram.

GridLM leverages game telemetry data, processes it through a structured pipeline, and uses a large language model (LLM) to generate insights for player improvement. Telemetry data is transmitted by the game in the form of UDP packets, which are directed to a Node.js server running locally. The server is configured to listen on a specific port and handle incoming data in real time. Upon receiving the data, the server parses each UDP packet to extract relevant telemetry metrics, processes and transforms the data as needed, and saves the processed data from the player's best lap into a local JSON file at the end of the session.

Once the session concludes, the JSON file is loaded into memory for further processing. Timestamp strings are converted to date objects, and the processed data is inserted into a MongoDB time-series collection hosted on MongoDB Atlas using Mongoose. The use of a time-series collection ensures efficient storage and querying of telemetry data, making it ideal for handling time-dependent metrics in a cloud-hosted environment.

To analyze the telemetry data, a Python script is manually triggered to retrieve data from the MongoDB database via the MongoClient library. This script calculates rolling averages for key metrics and compares them against predefined baseline averages to identify percentage differences. These differences quantify the gaps between the player's performance and an established baseline, forming the basis for actionable insights.

The processed comparison data is then formatted into a prompt template using LangChain. This template, enriched with the player's performance metrics, is used to invoke an LLM, which could be GPT-4o mini, Llama 3.2 3b, or Llama 3.2 3b-Instruct. The LLM generates personalized coaching suggestions based on the input data, providing actionable feedback to the player. Interactions with the LLM are tracked and logged using LangSmith, ensuring traceability and enabling further analysis of the system's performance.

Finally, the LLM's response is saved locally as a markdown file, creating a persistent record of the coaching insights. By integrating real-time data processing, statistical analysis, and advanced natural language processing, this system delivers a scalable and efficient solution for enhancing player performance in *Forza Motorsport (2023)*[1].

[1]The source code is available online at https://github.com/KShervington/GridLM-ForzaMS23

$$S = \left\lfloor \frac{D}{\frac{L}{N}} \right\rfloor + 1$$

S : Segment number
D : Distance traveled in the current lap
L : Total length of the track
N : Number of segments in the track

(1)

Figure 1: Equation to calculate the segment of track where a telemetry data point was recorded

3.2 Telemetry Data Acquisition

The first step in data acquisition was to ensure telemetry data from the game was accessible. In order to acquire the required data, Forza Motorsport's "Data Out" feature was used. When enabled, telemetry data is sent to a predetermined IP address at a rate of 60 UDP packets per second. The feature was configured, via in-game menus, to send data to 127.0.0.1 (i.e. localhost) over port 3000, which is the IP-port combination on which a local Node.js server was run. Forza offers two packet formats: "Sled" or "Dash"; the latter was chosen due to the additional data it offers [11].

To listen for incoming packets, the `dgram` module was used to create a web socket that could receive UDP packets. Whenever a packet is received ("message" in `dgram` terms), a function is triggered to parse the packet and extract specific points of data based on their position within the packet. Although the data positions can be calculated from documentation provided by the game's developers[11], an existing telemetry application was referenced to see the positions of data within a packet, along with their respective types [2].

3.3 Processing and Storage

The UDP packets were parsed to extract key metrics relevant to gameplay performance, such as speed, throttle percentage, braking force, and vehicle geometry. Parsed data was processed to add derived metrics, and organized into a JSON object. Some notable processing is as follows:

- **Timestamp Conversion**: Converted from 32-bit to 64-bit integers for compatibility.
- **Throttle Adjustment**: Rescaled from a 0-255 range to a more intuitive 0-100 range.
- **Current Lap Distance**: Distance from the telemetry data was cumulative so the distance traveled in a given lap had to be calculated by subtracting the product of the current lap number and total track length from the cumulative distance driven.
- **Lap Segmentation**: Custom lap segments were created to facilitate data analysis as the original data lacked inherent segment data. The current segment number of any given telemetry data object was calculated using equation 1. In that equation, total length of the track was a hard-coded constant which was determined by manually analyzing distance data from the game's telemetry output.

While the server was running, telemetry data was ingested and kept in memory for the duration of each lap. Upon completion of a lap, that data was pushed to a separate object which tracked data on the best lap of a session. This process would continue until a server shutdown gets initiated, at which point, a series of operations were performed:

(1) "Best Lap" data written to a local JSON file.
(2) JSON data loaded from local file into memory
(3) Data timestamps converted from strings to datetime values
(4) Insert all data into MongoDB time-series collection
(5) Disconnect from database
(6) Shutdown Node.js server

Data was stored in a cloud-hosted cluster via MongoDB Atlas. As opposed to collections which store data as documents, our telemetry data was stored in time series collections. The time-series structure is ideal for the chronological nature of our data and, as such, simplifies queries for data processing and analysis[8]. Within our GridLM cluster, we created a database for the track on which we tested. Two collections existed within that database, one for player telemetries, and another for reference telemetries, recorded by a high-performing player.

3.4 Data Retrieval and Preparation for Analysis

In preparation for model inference, data was retrieved one segment at a time. This was appropriate because only one lap worth of data was stored in each of our two collections. Data retrieval was triggered by manually running a python script. For simplicity, the same script also handled Langchain interactions, which will be discussed later. Each retrieved segment was further broken down into sub-segments. We calculate averages for key metrics, like speed, brake pressure, acceleration, and throttle percentage using **windowed averaging**. The process works as follows:

- Divide lap segment data into n sub-segments. We used an n of 10.
- Create a window, appropriately sized to the number of data points in each sub-segment.
- Use the window to calculate rolling averages for each metric over the segment. The result is a simplified array of averaged values that minimizes data volume while attempting to preserve essential telemetry patterns and insights.
- The x and y components of acceleration were combined before the rolling average was calculated.

For telemetry data on gear values, the rolling mode within each sub-segment was calculated instead of the rolling average. The calculated rolling averages for player and reference telemetries get stored in their respective arrays. This was the first, and most significant, step in downsizing telemetry data. Each segment had roughly 300 data points, based on the roughly 3000 data points. The described downsizing operations reduced those 300 data points into just 10 values for each metric, in each segment. All equations can be found in Appendix B.

However, feeding the remaining data proved to be very inefficient in LLM token usage. Therefore, we computed percentage differences between player and reference telemetries. This served to represent how the performance of the player compared to the reference player data. These comparisons were encoded in one of three ways:

- "x% higher"
- "x% lower"
- "identical"

The intuition behind encoding the comparisons this way was to introduce some semblance of natural language so as to improve the LLM's ability to interpret information. See Appendix C Listing 3 for an example of the encoded data which was injected into our prompt templates.

3.5 LangChain Integration and Prompt Design

To facilitate interaction with the LLM, we integrated **LangChain**, a framework to manage interactions with LLMs. Using LangChain, we designed prompt templates that organize downsized telemetry data logically for interpretation.

Prompt Template Development:

- A **system message** provided the LLM with context, instructing it to analyze telemetry data and offer actionable insights on speed, braking, and throttle from the perspective of an expert driving coach.
- The **human message** outlined the two main tasks for the LLM to perform (rate the driver's performance and provide a natural language assessment). This prompt also contained the structure that the LLM was meant to follow for its response, as well as an example of a potential assessment it could create. Variables, such as track name, and the player-reference comparison data were injected into this prompt (Listing 2 in Appendix C).

All prompt templates were created using LangChain's 'PromptTemplate' package, which enabled the injection of dynamic information, like comparison data and database name [7]. The latter was the same as the track, to give the LLM added context. The formatted prompt templates were used to invoke a response from our chosen LLM. The entire process can be seen in Figure 2.

3.6 LLM Selection and Invocation

In the development of GridLM, three models were used and evaluated: 1) Llama3.2 3B, 2) Llama3.2 3B Instruct, and 3) GPT 4o-mini-2024-07-18. The two Llama models were run locally and invoked via LangChain's Ollama integration[6]. GPT 4o-mini responses were acquired from the OpenAI API via LangChain's ChatOpenAI python library. The Llama models were chosen to evaluate the effectiveness of the models compared to the industry standard offered by OpenAI. Additionally, comparisons could take place between the base Llama3.2 3B and its instruction-tuned variant. The 3B versions were used because they were assumed to be superior to the 1B version, and the multi-modal capabilities of higher parameter Llama models were not needed[1].

All models were configured with a temperature value of 0.5 to minimize hallucination while maintaining enough creativity to allow the models occasionally provide nuanced coaching instructions which extrapolated beyond the given data. Listing 1 shows the entire configuration of the Llama models. While the GPT model ran on OpenAI's servers, the Llama models were run on an Nvidia GeForce GTX 1080 graphics card with 8GB of VRAM.

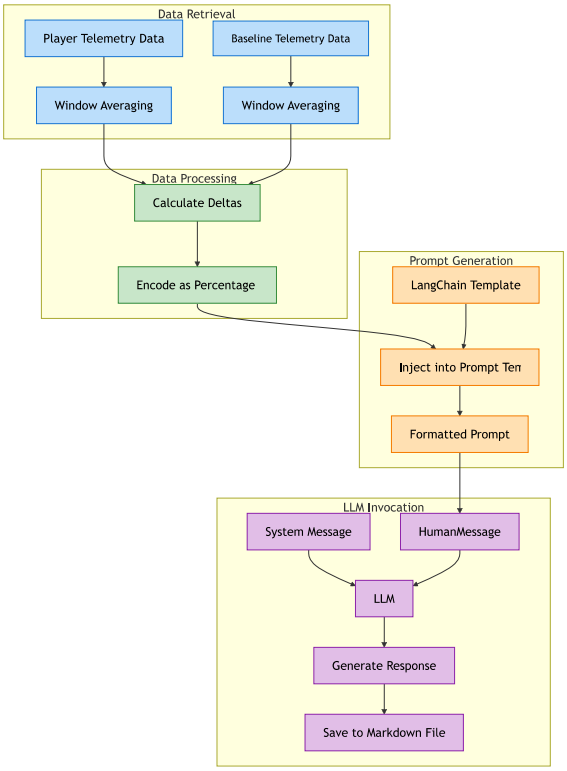

Figure 2: Flowchart of Prompt Creation & Response Generation, which illustrates the operations involved in generating a response from an LLM using telemtry data

```
MODEL = ChatOllama(
    disable_streaming=True,
    temperature=0.5,
    model='', # Ollama model name
    num_ctx=10000,
    num_gpu=1,
    num_predict=5000,
    # number of physical cores
    num_thread=multiprocessing.cpu_count() / 2,
)
```

Listing 1: Llama Model Configuration

3.7 LLM Response Handling and Experiment Design

After being invoked, LLM responses were stored in a local markdown file for persistence. Alternatively, invocation information, including responses, could be accessed via LangSmith, which tracked various metrics related to interactions with LLMs while using the LangChain framework. Local markdown files were formatted using Visual Studio Code's Prettier extension. See Appendix C Listing 4 for an example of a response from GridLM.

In order to evaluate model performance and responses, the model invocation logic was modified to iteratively create files containing LLM responses in a predetermined number of files. Using the same prompt and telemetry data, each model generated eight separate responses. The only requirement with generated responses is that it needed to be an even number so that each responses could be compared against others generated by the same model. With a total of 24 responses, we designed a script that take the two models defined by the experimenter and ran QuestEval on the responses of the two models.

The QuestEval task used to compare responses was the default `text2text` task, which measures the similarity between two English texts. Notably, QuestEval measures the semantic similarity of texts [10]. Therefore, by comparing the first four of a model's responses to the last four responses it generated, we can compute the relative consistency of the model in performing tasks required as part of GridLM. We gathered QuestEval scores for our prompt with an assessment example (one-shot prompt), and without. Additionally, we collected performance data from LangSmith to perform our evaluations.

4 RESULTS

Model	Llama3.2-3b	Llama3.2-3b-instruct	GPT-4o-mini-2024-07-18
Llama3.2-3b	**0.4062455107**	0.3745652383	0.3797228415
Llama3.2-3b-instruct	0.4333609465	**0.4084527814**	0.4061626924
GPT-4o-mini-2024-07-18	0.3954983227	0.3625729393	**0.4535591706**
Baseline	0.8729		

Table 1: Model Consistency (Standard Prompt) (temperature = 0.5)
QuestEval similarity scores from comparing a model's first four responses (left column) to the last four responses (top row) from itself or another model.

Table 1 shows the similarity scores, generated by QuestEval, for each model combination. The highlighted values show "self-consistency" for each model. Higher values indicate more semantic consistency across model responses. GPT 4o-mini proved to be substantially more consistent than both Llama models. The consistency difference between the two Llama models, however, was not significant. To put the stated values into perspective, refer to the baseline value, which is the similarity score QuestEval outputted when it was given two identical strings to compare. The evaluation results shown in Table 1 were created using our standard prompt, which did not include an assessment example. The remaining evaluations were done with a one-shot prompt.

Table 2 demonstrates model consistency when an example assessment was provided as part of our prompt. Compared to the scores seen in Table 1, we can see that self-consistency improved significantly for both Llama models. This indicates that these models respond well to few-shot prompting. Interestingly, GPT 4o-mini's self-consistency actually decreased compared to standard prompting. Although the difference isn't significant enough to draw a confident conclusion, this score decrease could imply that GPT performance is degraded as a result of few-shot prompting, in the context of the GridLM system.

Model	Llama3.2-3b	Llama3.2-3b-instruct	GPT-4o-mini-2024-07-18
Llama3.2-3b	**0.4982744081**	0.430323257	0.4118958818
Llama3.2-3b-instruct	0.423977187	**0.4741555659**	0.3970852224
GPT-4o-mini-2024-07-18	0.3920362059	0.39620884623	**0.4310150017**
Baseline	0.8729		

Table 2: Model Consistency (temperature = 0.5)
QuestEval similarity scores from comparing a model's first four responses (left column) to the last four responses (top row) from itself or another model.

The graph in Figure 3 illustrates the number of hallucinations generated by the three models—llama3.2-3b-instruct, llama3.2-3b, and gpt-4o-mini-2024-07-18—across different responses. Hallucinations occur when a model provides output that includes unsupported or irrelevant information. In this evaluation, llama3.2-3b demonstrated the poorest performance, showing significantly more hallucinations compared to the other models, particularly in responses 2 and 6. This spike highlights the model's struggle to maintain accuracy and relevance during certain tasks.

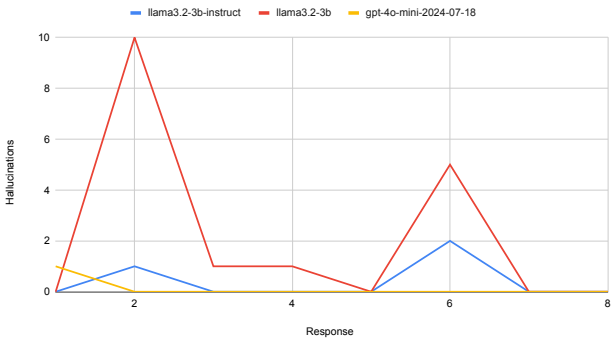

Figure 3: Model Hallucinations
Number of hallucinations in each model's responses. A hallucination was a clear mention or suggestion about information that was not given to the model as context

Conversely, llama3.2-3b-instruct and gpt-4o-mini-2024-07-18 perform far better in this metric, with minimal or no hallucinations across most responses. gpt-4o-mini-2024-07-18, in particular, exhibits exceptional reliability, maintaining zero hallucinations throughout nearly all responses, which reflects its ability to generate accurate and contextually relevant outputs. The results clearly identify llama3.2-3b as the least consistent and reliable model in managing hallucinations, indicating a need for significant improvements in this area.

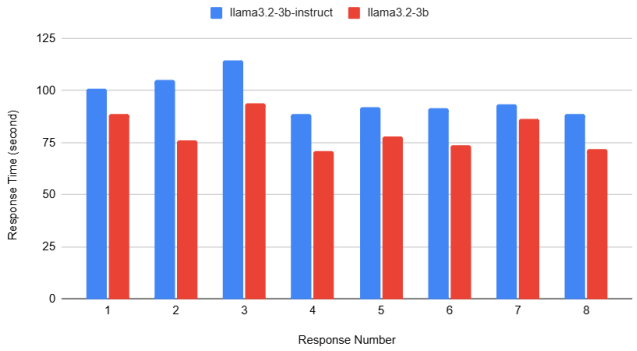

Figure 4: Latency
Number of seconds after a model receives a prompt for it to output a response

Figure 4 presents the latency of the models (i.e. the number of seconds each model took to generate a response after being given a prompt). Latency metrics from `gpt-4o-mini-2024-07-18` were excluded because its latency, being that it was being accessed via API, was not comparable to that of the Llama models which were running locally. Generally, the GPT's latency times were roughly 10 seconds. Between the Llama models, llama3.2-3b demonstrated consistently faster response times than llama3.2-3b-instruct.

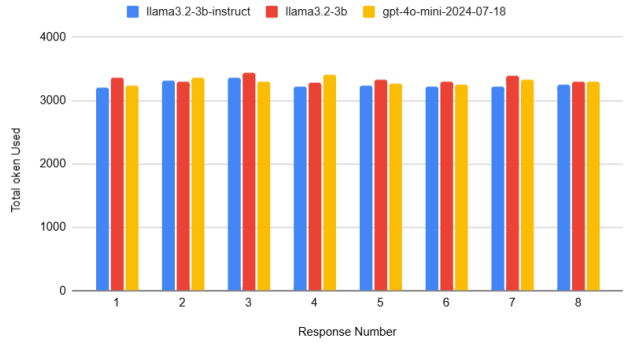

Figure 5: Token Usage
Total number of tokens used in each LLM interaction. This includes tokens from prompts, as well as responses

Token usage provides an essential metric for evaluating the efficiency of models in generating responses. The data reveals that `llama3.2-3b` consistently uses more tokens compared to llama3.2-3b-instruct and `gpt-4o-mini-2024-07-18` across all responses. This indicates that the base version of the Llama model is less efficient in structuring concise outputs. `gpt-4o-mini-2024-07-18`, while slightly more token-efficient, still consumed more tokens than llama3.2-3b-instruct in most cases.

`llama3.2-3b-instruct` demonstrated a consistently lower token usage, reflecting its ability to generate more concise responses. The close range of token usage across all responses for each model suggests stability in token consumption, but the overall efficiency of `llama3.2-3b-instruct` makes it the most token-efficient model among the three.

Comparing `llama3.2-3b-instruct`'s 26,044 tokens used across all 8 responses to `llama3.2-3b`'s 26,687, we can see that the base variant used approximately 80 more tokens per response. It is important to note that token efficiency is not correlated with response quality. This was found to be true when manually reviewing `llama3.2-3b-instruct`'s responses which were often missing numerical performance assessments and used vague wording which, although concise, would not have been regarded as helpful coaching suggestions.

5 CONCLUSION AND FUTURE WORK

The GridLM project presented a novel approach to integrating telemetry data with large language models (LLMs) to provide personalized gameplay coaching for Forza Motorsport. By utilizing advanced data processing frameworks, including MongoDB for data storage and LangChain for LLM integration, the system was designed to analyze telemetry metrics such as speed, throttle, and braking, generating actionable insights to support player performance improvement.

Through the development and evaluation phases, the system demonstrated its ability to generate coherent responses and deliver targeted coaching feedback. Challenges such as LLM hallucinations and token usage inefficiencies were identified and addressed through optimization strategies. The creation of GridLM achieved the objective of developing a prototype capable of bridging the skill gap between casual and competitive players. This work provides a foundation for future advancements, including the integration of more extensive telemetry metrics, real-time processing capabilities, and enhancements to LLM efficiency and reliability to further improve the player experience.

Future work can expand on this foundation by integrating advanced multimodal LLMs capable of processing diverse data inputs, such as gameplay footage, to provide more comprehensive coaching feedback. Extending the system to other racing games and simulations would validate its versatility across different platforms. Collaborations with AI coaching platforms like trophi.ai could offer valuable insights to enhance coaching effectiveness. Additionally, incorporating real-time feedback and adaptive learning mechanisms would enable dynamic, personalized coaching tailored to individual player needs, further advancing AI-driven systems in racing simulations.

REFERENCES

[1] AI, M. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, 2024. Accessed: 2024-10-02.

[2] Baccus, A. Forza telemetry, 2021. Accessed: 2024-09-30.

[3] Gallotta, R., Todd, G., Zammit, M., Earle, S., Liapis, A., Togelius, J., and Yannakakis, G. N. Large language models and games: A survey and roadmap. *arXiv preprint arXiv:2402.18659* (2024).

[4] Hu, S., Huang, T., Ilhan, F., Tekin, S., Liu, G., Kompella, R., and Liu, L. A survey on large language model-based game agents. *arXiv preprint arXiv:2404.02039* (2024).

[5] Lai, J. The neverending game: How ai will create a new category of games. *Andreessen Horowitz* (2023).

[6] LangChain. Chatollama, 2024. Accessed: 2024-12-03.

[7] LangChain. Prompt templates, 2024. Accessed: 2024-12-03.

[8] MongoDB, I. Mongodb time series. https://www.mongodb.com/products/capabilities/time-series. Accessed: 2024-12-02.

[9] Multiplatform AI. Forza motorsport transforms racing simulation with advanced machine learning techniques. *Multiplatform AI* (2024).

[10] Rebuffel, C., Scialom, T., Soulier, L., Piwowarski, B., Lamprier, S., Staiano, J., Scoutheeten, G., and Gallinari, P. Data-questeval: A referenceless metric for data to text semantic evaluation. *arXiv preprint arXiv:2104.07555* (2021).

[11] Support, F. Forza motorsport "data out" documentation, 2023. Accessed: 2024-10-02.

Appendices

A

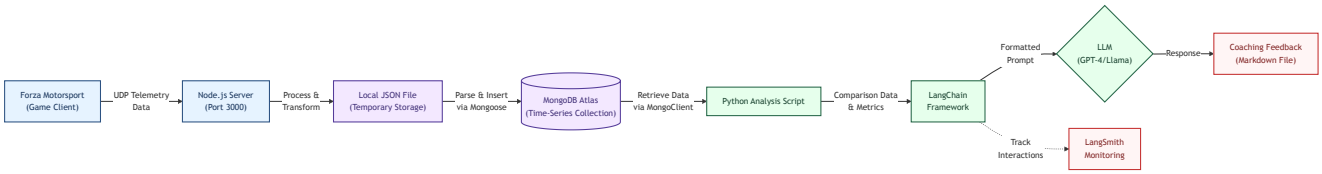

Figure 6: Architecture Diagram of GridLM

B

GENERALIZED DEFINITIONS

For each segment $s = 1, 2, \ldots, S$:

$$N_s = \text{Number of data points in segment } s$$

$$K = 10$$

$$w_s = \left\lfloor \frac{N_s}{K} \right\rfloor + 1$$

$$K_s = \left\lceil \frac{N_s}{w_s} \right\rceil$$

For each window $k = 1, 2, \ldots, K_s$:

$$I_{s,k} = \{\, i \mid i = (k-1)w_s + 1, \ldots, \min(kw_s, N_s) \,\}$$

COMPUTATION OF COMBINED ACCELERATION

If x represents acceleration:

$$x_{s,i} = \sqrt{(\text{accelerationX}_{s,i})^2 + (\text{accelerationY}_{s,i})^2}$$

COMPUTE ROLLING AVERAGE FOR x

$$\overline{x}_{s,k} = \frac{1}{|I_{s,k}|} \sum_{i \in I_{s,k}} x_{s,i}$$

MOST COMMON GEAR VALUE

$$\text{mode\_gear}_{s,k} = \text{mode}\left(\{\, \text{gear}_{s,i} \mid i \in I_{s,k} \,\}\right)$$

METRICS REPRESENTED BY x

$$x \in \{\text{speed},\ \text{brake pressure},\ \text{throttle percentage},\ \text{combined acceleration}\}$$

C

Listing 2: Human Message Prompt Template

```
"""
Below are summarized telemetry data for the track {track_name}, segmented for analysis. Your task is to:
1. Performance Rating (out of 10): For each segment, rate the driver's performance based on how their telemetry data
    compares to the baseline data.
2. Assessment: Write a concise assessment (maximum 50 words) for each segment, highlighting strengths, weaknesses, and
    potential improvements.

Please follow this structure for your response:
# Track Name: {track_name}


## Segment <segment number>
- **Performance Rating (out of 10)**:
- **Assessment**:


## Segment <next segment number>
- **Performance Rating (out of 10)**:
- **Assessment**:


Ensure that your response is consistent and structured. Be specific and actionable in your assessments,
identifying key changes that could lead to improved performance. Here is an example of an assessment:
Minimize braking in this segment, focusing on maintaining momentum. Apply higher brake pressure for sharper braking. Try
holding 3rd gear through this segment to carry more speed. Apply throttle earlier to improve acceleration.


Repeat the defined structure for all 10 segments in the data below.


Comparison of Driver Telemetry Data to the Baseline Telemetry Data:
{data_comparisons}
"""
```

Listing 3: Excerpt of Comparison Data Injected into Prompt Template

```
{
  "Segment 1": {
    "avg_speed_delta_over_time": "3% lower, 2% lower, 2% lower, 2% lower, 2% lower, 2% lower, 2% lower, 2% lower, 2% lower",
    "avg_acceleration_delta_over_time": "17% lower, 10% higher, 17% higher, 141% higher, 82% higher, 72% lower, 3% lower, 125% higher, 58% lower",
    "avg_brake_pressure_delta_over_time": "identical, identical, identical, identical, identical, identical, identical, identical, identical",
    "avg_throttle_percentage_delta_over_time": "identical, identical, identical, identical, identical, identical, identical, identical, identical",
    "most_common_gear_delta_over_time": "identical, identical, identical, identical, identical, identical, identical, identical, identical"
  },
  "Segment 2": {
    "avg_speed_delta_over_time": "2% lower, 2% lower, 2% lower, 2% lower, 8% lower, 10% lower, 12% lower, 15% lower, 19% lower",
    "avg_acceleration_delta_over_time": "41% lower, 24% higher, 1% higher, 103% higher, 25% lower, 13% lower, 22% lower, 12% lower, 25% lower",
    "avg_brake_pressure_delta_over_time": "identical, identical, identical, 94.83 higher, 37% higher, identical, identical, identical, 500% higher",
    "avg_throttle_percentage_delta_over_time": "identical, identical, identical, 30% lower, 100% lower, identical, identical, identical, 28.7 higher",
    "most_common_gear_delta_over_time": "identical, identical, identical, identical, identical, identical, 1 gear(s) lower, 7 gear(s) higher, 1 gear(s) lower"
  },
  "Segment 3": {
    "avg_speed_delta_over_time": "12% lower, 0% higher, 2% lower, 1% higher, 1% higher, 1% lower, 2% lower, 3% lower, 5% lower",
    "avg_acceleration_delta_over_time": "19% lower, 10% lower, 18% lower, 5% lower, 2% lower, 3% lower, 3% lower, 10% lower, 2% higher",
    "avg_brake_pressure_delta_over_time": "100% lower, 20% lower, 4% lower, identical, identical, identical, identical, identical, identical",
    "avg_throttle_percentage_delta_over_time": "100.0 higher, 38.45 higher, 55.68 higher, 15.04 higher, 17.72 higher, 46.82 higher, 305% higher, 10% lower, 16% lower",
    "most_common_gear_delta_over_time": "1 gear(s) lower, identical, identical, identical, identical, identical, identical, identical, identical"
  },

  ...

  "Segment 10": {
    "avg_speed_delta_over_time": "0% higher, 0% higher, 0% higher, 0% lower, 1% lower, 1% lower, 1% lower, 1% lower, 1% lower",
    "avg_acceleration_delta_over_time": "64% lower, 76% lower, 43% lower, 28% lower, 558% higher, 20% lower, 12% higher, 4% lower, 76% lower",
    "avg_brake_pressure_delta_over_time": "identical, identical, identical, identical, identical, identical, identical, identical, identical",
    "avg_throttle_percentage_delta_over_time": "identical, identical, identical, identical, identical, identical, identical, identical, identical",
    "most_common_gear_delta_over_time": "identical, identical, identical, 1 gear(s) higher, 1 gear(s) higher, 1 gear(s) higher, 1 gear(s) higher, 1 gear(s) higher, identical"
  }
}
```

Listing 4: Excerpt of Response from `gpt-4o-mini-2024-07-18`

```
# Track Name: SuzukaCircuit

## Segment 1
- **Performance Rating (out of 10)**: 6
- **Assessment**: Consistent brake pressure is good, but overall speed is down. Focus on improving acceleration in the latter part of the segment. Aim to reduce throttle lag and maintain higher speeds through corners.

## Segment 2
- **Performance Rating (out of 10)**: 5
- **Assessment**: Speed drops significantly, particularly in the second half. Increase throttle earlier after braking and consider shifting up sooner to maintain momentum. Work on maintaining consistent acceleration throughout.

## Segment 3
- **Performance Rating (out of 10)**: 4
- **Assessment**: Notable speed and acceleration loss. Focus on optimizing braking points and minimizing braking duration. Experiment with gear selection to find the optimal balance for speed and acceleration.

...

## Segment 10
- **Performance Rating (out of 10)**: 6
- **Assessment**: Speed is relatively stable, but acceleration shows variability. Focus on achieving consistent throttle application and optimizing gear changes to enhance overall performance in this segment.
```