

Graphical Abstract

Large Language Models as Autonomous Spacecraft Operators in Kerbal Space Program

Alejandro Carrasco, Victor Rodriguez-Fernandez, Richard Linares

Large Language Models as Autonomous Spacecraft Operators in Kerbal Space Program

Alejandro Carrasco^a, Victor Rodriguez-Fernandez^{a,b}, Richard Linares^a

^a*Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 77 Massachusetts Avenue, Cambridge, MA, 02139, USA*

^b*Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingeniería de Sistemas Informáticos, Madrid, 28031, Madrid, Spain*

Abstract

Recent trends are emerging in the use of Large Language Models (LLMs) as autonomous agents that take actions based on the content of the user text prompts. We intend to apply these concepts to the field of Control in space, enabling LLMs to play a significant role in the decision-making process for autonomous satellite operations. As a first step towards this goal, we have developed a pure LLM-based solution for the Kerbal Space Program Differential Games (KSPDG) challenge, a public software design competition where participants create autonomous agents for maneuvering satellites involved in non-cooperative space operations, running on the KSP game engine. Our approach leverages prompt engineering, few-shot prompting, and fine-tuning techniques to create an effective LLM-based agent that ranked 2nd in the competition. To the best of our knowledge, this work pioneers the integration of LLM agents into space research. The project comprises several open repositories to facilitate replication and further research. The codebase is accessible on GitHub, while the trained models and datasets are available on Hugging Face. Additionally, experiment tracking and detailed results can be reviewed on Weights & Biases.

Keywords: Large Language Models, Autonomous Agents, Kerbal Space Program, Prompt Engineering, Fine-tuning, Spacecraft Control

1. Introduction

Large Language Models (LLMs) are, without a doubt, the last major breakthrough in the evolution of artificial intelligence systems. Since the release of ChatGPT OpenAI (2022) at the end of 2022, we have seen a plethora of applications and use cases emerge across various industries. From generating human-like text to

aiding in code completion, LLMs have significantly impacted the way we interact with technology and the possibilities of what AI can achieve.

In recent times, LLMs have progressed from text-based applications to *language agents* capable of taking context-aware actions, as demonstrated by Anthropic’s computer use agent¹. By leveraging contextual information, LLMs can autonomously perform tasks which usually correspond to software-based systems. Expanding on this, these so-called agents extend beyond software applications, as researchers are applying this concept to physical systems such as LLM-driven robots Wang et al. (2023) and self-driving car motion planners Mao et al. (2023).

Historically, spacecraft Guidance, Navigation, and Control (GNC) systems have relied on classical control methods like Proportional-Integral-Derivative (PID) control, Linear Quadratic Regulators (LQR), and more advanced techniques such as Model Predictive Control (MPC) and robust control. While these methods have ensured reliable spacecraft operations, they often require extensive tuning and precise dynamic modeling. AI and machine learning, particularly Reinforcement Learning (RL), have emerged as promising alternatives, offering more adaptive solutions that optimize control policies through environmental interaction, with notable examples including agents trained for tasks such as sensor-tasking Siew et al. (2022) and planetary landing Gaudet et al. (2020).

This work focuses on space applications, particularly the development of autonomous agents for the guidance and control of spacecraft. However, unlike other fields of AI research, the space domain suffers from a lack of publicly available simulation environments, which are essential for training AI agents in complex scenarios and benchmarking autonomous control methods. To address the lack of simulation environments in space applications, Allen et al. introduced *SpaceGym* Allen et al. (2023), which includes the Kerbal Space Program Differential Games suite (KSPDG)—the primary focus of this work.

However, KSPDG is unsuitable for RL training due to technical and design constraints. The KSP engine lacks capabilities for parallel, accelerated, and headless operations necessary for faster-than-real-time RL training. Moreover, KSPDG’s creators intentionally designed it as an evaluation framework, a “true test set” environment that minimizes overfitting and prioritizes unbiased testing of AI agent capabilities, diverging from RL’s iterative training paradigm. In contrast with traditional RL, which often struggles with sample inefficiency and the need for explicit reward functions, LLMs offer a promising alternative. Studies have shown that advanced LLMs, like GPT-4, can outperform state-of-the-art RL algorithms in complex

¹<https://www.anthropic.com/news/3-5-models-and-computer-use>

games simply by analyzing academic texts and reasoning Wu et al. (2023), achieving sophisticated trajectories and strong zero-shot performance.

To overcome the limitations of RL in creating autonomous agents for environments such as KSDPG, as well as for other space operations where numerous simulated data cannot be provided, we propose to adapt the current trend of LLM-based agents to develop an “intelligent” operator that controls a spacecraft based on the real-time telemetry of the environment, relying solely on natural language for both input and output. As depicted in both figures in fig. 1, we design the classic RL loop by interfacing the simulation environment (KSDPG) with a LLM, transforming the real-time observations (or state) of the mission as textual user prompts that are fed to the model. The LLM then processes the prompt and replies with an action that will be plugged in KSDPG to control the spacecraft. In this work, we leverage GPT-3.5 and LLaMA models to develop an intelligent agent capable of controlling a spacecraft in real time. We explored various strategies to present the mission environment observations as textual inputs to the models and refined their performance using data collected from human gameplay and expert-mimicking bots. In both editions of KSPDG, our LLM centered agents were the only non-traditional optimization method to reach the podium, consistently advancing to the finals and securing second place consecutively ².

When deciding on an LLM approach, there are three primary strategies: creating a new model, prompt-engineering to leverage the capabilities of an existing model, or fine-tuning a model to optimize performance. Given the technical and resource constraints of this work, creating a new LLM is not feasible, though such a requirement is rare in practice. Instead, this study focuses on leveraging existing models through prompt-engineering and fine-tuning, which serve as viable and efficient alternatives. Fine-tuning, in particular, offers the opportunity to address specific limitations of base models, such as response latency or consistency, enabling the refinement of models to ensure they perform optimally for targeted tasks.

The LLM landscape has evolved rapidly over the past two years, with significant advancements and frequent updates leading to the emergence of models that rival the once-dominant ChatGPT. Notable examples include Claude Anthropic (2023), Gemini DeepMind (2023), and open-source models like Mistral Dettmers et al. (2023), DeepSeek DeepSeek-AI (2024) and LLaMA Touvron et al. (2023). This study uses GPT for its ease of use and LLaMA for its community support and open-source flexibility.. As illustrated in fig. 1, two strategies are compared: one involving prompt-engineering, and another using fine-tuned models.

²Kerbal Space Program Differential Game Challenge

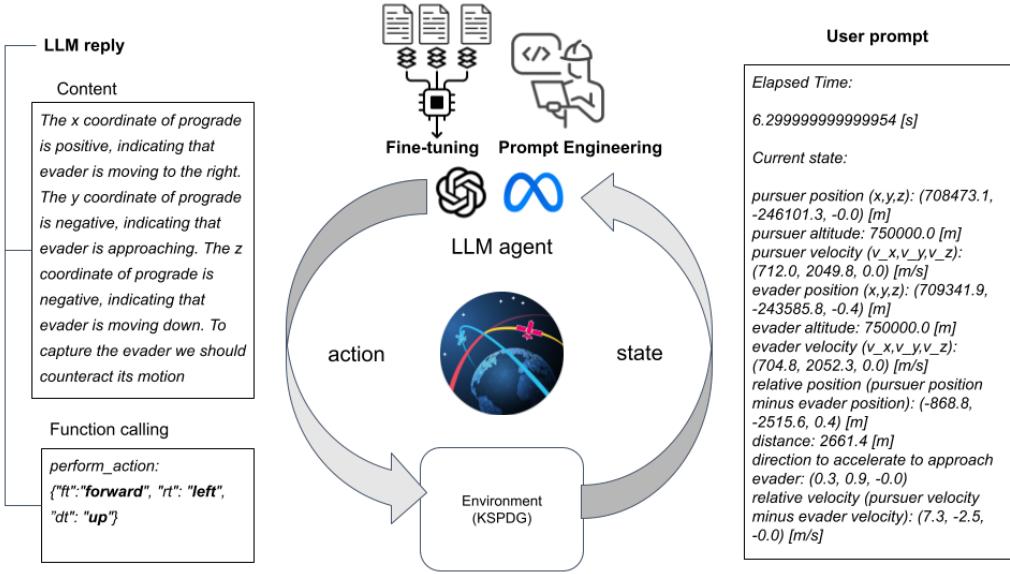


Figure 1: Overview of both LLM strategies based on the KSPDG Challenge agent implementation.

2. Large Language Models (LLMs)

The 21st century has seen an explosion of interest and progress in AI (Computer Vision, Sentiment Analysis, Natural Language Processing). However, the development of large language models (LLMs) has been unprecedented. OpenAI’s GPT-3 Brown et al. (2020), released in 2020, demonstrated remarkable capabilities in generating human-like text and performing complex tasks based on textual input.

LLMs like GPT-4 OpenAI (2023), Claude Anthropic (2023), Gemini DeepMind (2023), Mistral Dettmers et al. (2023), and LLaMA Touvron et al. (2023) have further pushed the boundaries of what AI can achieve. These models leverage vast amounts of data and computational power, resulting in great performance across a range of applications, from writing essays to generating code. The field continues to evolve rapidly, with ongoing research and development aimed at enhancing the capabilities and applications of AI Arslanian and Fischer (2019).

Large language models are designed to understand and generate human language by processing massive datasets containing diverse text from the internet, books, and other sources. They employ deep learning techniques, particularly transformers Vaswani et al. (2017), which allow them to manage context over long passages of text effectively. This ability to maintain context makes them proficient in tasks such as translation, summarization, or question answering.

Moreover, LLMs have shown remarkable zero-shot, one-shot, and few-shot learn-

ing capabilities Brown et al. (2020). This means they can perform tasks they were not explicitly trained on, given minimal examples or instructions. For instance, GPT-3 can write poetry, generate code, and even perform some reasoning tasks with little to no specific training.

2.1. Techniques for LLMs

LLMs leverage a variety of techniques to enhance their performance and usability. Some of the most notable techniques include:

Prompt Engineering: Crafting precise prompts guides the LLM in generating desired responses. Even slight variations in prompts can significantly influence the model’s behavior and output, tailoring it to specific needs. Prompt engineering is essential for optimizing LLM performance across various applications Reynolds and McDonell (2021).

Chain of Thought: This technique guides the model to generate logical steps leading to the final answer, breaking down complex problems for better reasoning and interpretability Wei et al. (2022). It has evolved from user-driven to model-driven CoT, demonstrating remarkable capabilities. Combined with technical ingenuity, this shift has enabled models like DeepSeek DeepSeek-AI (2024) to achieve outstanding performance.

Few-Shot Prompting: Generalizing from a small number of examples provided within the input prompt. This technique enables the model to infer patterns and apply them to new, unseen data Brown et al. (2020).

Fine-Tuning: Training a pre-trained LLM on a specific dataset related to a particular task or domain. This adapts the LLM to specialized tasks by learning the nuances and specifics of the target domain Howard and Ruder (2018).

Function Calling: This technique leverages a structured JSON schema, allowing LLMs to interact with external codebases in an API-like manner. By specifying tools or functions in JSON format, users can instruct the model to execute tasks or access external data sources. For example, retrieving weather data might involve a function call like:

Task: Retrieve weather data for London using the function call.

Output:
{
 "function": "getWeatherData", "parameters": { "location": "London" }
}

2.2. Limitations of LLMs

The biggest challenge facing LLMs is a phenomenon known as hallucinations. First introduced in the context of machine translation in 2018, the term refers to outputs that are entirely disconnected from the input, often incorrect or nonsensical Lee et al. (2018). Despite their advanced capabilities, LLMs occasionally produce responses that lack grounding in the provided input or real-world facts. This issue poses significant challenges for applications requiring high accuracy and reliability. Ongoing research aims to better understand the causes of hallucinations and develop effective strategies to mitigate them Ji et al. (2023).

At the same time, language has long been regarded as central to human reasoning, as argued by Wittgenstein and Piaget Wittgenstein (1953), Piaget (1926). Given this, the remarkable linguistic abilities of modern LLMs raise an important question: Could mastering language be the key to achieving artificial general intelligence? Motivated by this, this work explores pushing LLM capabilities beyond mere conversation.

3. Kerbal Space Program (KSP)

Kerbal Space Program (KSP)³ is a space flight simulation video game developed by the Mexican studio Squad, released in 2015. Although it is a game, it can be used as a simulation environment with the addition of *mods* that add new features such as more realistic physics.

The scope of KSP includes various spacecraft missions involving celestial bodies and maneuvering with goals such as escaping the atmosphere, reaching another planet, or intercepting a space object Engadget (2014). An interesting feature of this game is the ability to create missions that can be shared with other players. All these features convert KSP into a wonderful educational tool and simple work environment for initial testing of different space missions.

Although KSP does not offer a perfect simulation of reality, it has been praised for its accurate orbital mechanics, even forming a partnership with NASA that elevated its status beyond just a game⁴. The simulation environment is constrained to a two-body problem and is limited to a small number of planets, most commonly just an earth-shaped planet called Kerbin⁵.

³<https://store.privatedivision.com/game/buy-kerbal-space-program-ksp>

⁴<https://www.polygon.com/features/2014/1/27/5338438/kerbal-space-program>

⁵<https://kerbalspaceprogram.fandom.com/wiki/Kerbin>

3.1. Kerbal Space Program Differential Games

The KSPDG Challenge Allen (2023) facilitates a gymnasium and code for implementing agents into Kerbal Space Program (KSP). An agent is “a software entity that performs tasks autonomously by perceiving its environment through sensors and acting upon that environment through actuators” Russell and Norvig (2016).

In KSPDG, agents control a spacecraft’s movement in all three rotational axes (yaw, pitch, and roll) using the thrust engines. Actions are expressed in the spacecraft’s reference frame and include thrust magnitude for each axis and duration of the applied thrust. Although KSPDG allow numerical thrust magnitudes, the LLM agents presented in this study use discrete thrust values. KSPDG includes three scenarios:

- **Pursuer-evader (E1-E4):** The agent controls the pursuer. The main objective is to minimize the distance between the pursuer and the evader. For different scenarios in the Pursuer-Evader game, the evader’s initial orbit remained constant across all scenarios while the pursuer’s initial orbit varied. The pursuer and the evader have identical vehicle parameters. Participants were evaluated on metrics such as distance between pursuer and evader (m), speed at closest approach (m/s), pursuer fuel usage (kg), and time elapsed (s). The main differences between scenarios E1, E2, E3, and E4 lie in the evasive maneuvers implemented by the Evader vessel. This study focuses on E3 which employs a structured evasive strategy, where the Evader vessel activates full thrust to escape when the Pursuer is within a specified distance threshold.
- **Target Guarding (Lady-Bandit-Guard):** Agents control a bandit spacecraft to stay close to a lady spacecraft while avoiding a guard spacecraft. Scenarios vary based on initial orbits and vehicle capabilities that will not be covered in this study.
- **Sun-Blocking:** Agents aim to position a spacecraft between an evader and the sun. Scenarios vary according to initial orbits and evader strategies that will not be covered in this study.



Figure 2: Agent’s in-game viewpoint during a KSP mission.

3.2. Navigation

As illustrated in fig. 2, KSPDG provides a third-person perspective for navigation, incorporating multiple visual cues on a dashboard. The Navball, a spherical navigation display positioned at the bottom center of the screen, contains critical information necessary for in-spaceflight decision-making. A comprehensive explanation of in-space terminology and the markers used in KSP can be found in publicly available documentation (e.g., Kerbal Space Program Wiki (2018))

4. GPT as a Spacecraft Operator

The interaction with the Language Model (LLM) is primarily based on prompts. Prompts are the inputs provided to the model to elicit a desired response.

4.1. GPT Data

For their GPT models, OpenAI uses a specific fine-tuning format⁶. This format consists of a JSONL file with mandatory and optional parameters. Each entry includes a role: “user” for input prompts, “system” for system instructions, and “assistant” for model responses. This structure enables augmented chat generations for few-shot prompting Touvron et al. (2023).

KSPDG outputs *observations* for each state of the mission. These *observations* include the time elapsed (s), the vehicle mass (kg), the vehicle propellant (kg), positions for both pursuer and evader (x,y,z), as well as their velocities in the reference

⁶<https://platform.openai.com/docs/api-reference/chat>

frame of the celestial body⁷ (m/s). Some of these observations may be of any relevance or not considered by the LLM.

At the time of the KSPDG challenge, the OpenAI API did not support maintaining context between calls. To address this limitation, a sliding window with zero padding was used in some experiments. This technique involves prepending the user prompt with the last n conversations with the LLM where n is the window size using zero padding if required Beltagy et al. (2020).

Instead of using special tokens, some keywords were used to distinguish between the user and the assistant during chat interactions. Specifically, GPT identifies the user as *HUMAN* and the model as *ASISSTANT*.

4.2. Prompt Engineering

Prompt engineering involves designing the input text for better comprehension by the LLM. We systematically generate multiple variations, run simulations in KSPDG, and select the best-performing prompts. Optimal prompts include concise explanations of the system, state observations, and mission goals.

Basic prompt engineering does not address the arithmetic limitations of LLMs when handling large numbers⁸. Performing prompt augmentation, we enhance the observation space by computing the prograde vector to help the model achieve its mission.

The prograde vector is fundamental in orbital mechanics; it indicates the direction of movement of the spacecraft relative to a reference point. In the KSPDG environment, our spacecraft's nose always points toward the target, which serves as the reference for calculating the prograde. It is computed as follows:

$$\mathbf{p} = \mathbf{R}^{-1} \frac{\mathbf{v}_p - \mathbf{v}_e}{\|\mathbf{v}_p - \mathbf{v}_e\|} \quad (1)$$

where:

- **R**: Rotation matrix that transforms coordinates from the vessel to the celestial body reference frame.
- $\mathbf{v}_p, \mathbf{v}_e$: pursuer's and evader's velocity in the celestial body reference frame.

⁷The celestial body reference frame is a coordinate system centered on a celestial body (Kerbin) and rotates with it, while “vessel up” refers to the upwards direction in the vessel reference frame that is the coordinate system which is centered and oriented on the vessel under control

⁸This statement applies specifically to LLMs that have not been trained with explicit reasoning modules or reinforcement learning for arithmetic tasksDeepSeek-AI (2024), OpenAI (2024)

The Rotation Matrix \mathbf{R} is defined as:

$$\mathbf{R} = [\mathbf{e}_r \mathbf{e}_f \mathbf{e}_u] \quad (2)$$

where $\mathbf{e}_r, \mathbf{e}_f, \mathbf{e}_u$ are the unit vectors, in the reference frame of the celestial body, along the pitch (radial), roll (forward) and yaw (up) axes of the vessel.

$$\mathbf{e}_r = \mathbf{e}_f \times \mathbf{e}_u \quad (3)$$

$$\mathbf{e}_f = \frac{\mathbf{p}_e - \mathbf{p}_p}{\|\mathbf{p}_e - \mathbf{p}_p\|} \quad (4)$$

$$\mathbf{e}_u = \frac{\text{vessel_up}}{\|\text{vessel_up}\|} \quad (5)$$

where:

- $\mathbf{p}_p, \mathbf{p}_e$: pursuer's and evader's position in the celestial body reference frame.
- vessel_up : refers to the predefined orientation of the spacecraft's top side relative to the celestial body.

4.3. Few-shot Prompting

The LLM often fails to perform the function call correctly in its first response, causing subsequent errors. To mitigate this, we use a hand-made example, similar to a one-shot prompt Brown et al. (2020), including reasoning and a valid function call. This reduces erroneous function calls and improves subsequent responses.

In our previous work Rodriguez-Fernandez et al. (2024), we employed both a rule-based approach and Chain of Thought (CoT) prompting. While the rule-based approach improved performance, it significantly reduced generalization. In this study, we chose to retain only CoT, as it offers greater flexibility while maintaining strong performance.

Using the CoT approach, the quality of the output is heavily reliant on the model's capacity for reasoning. Therefore, we used the most advanced version available for the GPT-3.5 model⁹

Figure 3 introduces key terms (*prograde, reference frame, celestial body*), essential for understanding the user prompt. Figure 4 illustrates an example of a user prompt using CoT which was crafted from the experience of previous interactions.

⁹The specific model version used in this CoT study is gpt-3.5-turbo-0125, which was the latest model available.

You operate as an autonomous agent controlling a pursuit spacecraft. Your goal is to apply throttles to capture the evader given the positions and velocities of the pursuer and evader in celestial body reference frame and the direction of pursuer's velocity relative to evader or prograde. Throttles must be given in your vessel reference frame wherein the x axis points to the right, the y axis points towards the target and the z axis points upwards. The maximum throttle is 1. Reason step-by-step. After reasoning call the `perform_action` function.

Figure 3: System prompt used for eliciting Chain of Thought

USER: Given these observations *{KSPDG environment observations from a past action}*, what is the best throttle to capture evader?

ASSISTANT: The x coordinate of prograde is positive, indicating that pursuer is moving to the right. The y coordinate of prograde is negative, indicating that pursuer is approaching. The z coordinate of prograde is negative, indicating that pursuer is moving down. To capture the evader, we should move in the opposite direction in the x axis, towards the target in the y axis, and in the opposite direction in the z axis. This means we should apply throttles to move left, forward and up. Therefore we should call `perform_action({“ft”: “forward”, “rt”: “left”, “dt”: “up”})`.

Now answer the following question:

Given these observations *{Current KSPDG environment observations}*, what is the best throttle to capture evader?. **Reason step-by-step.**

Figure 4: Sample user prompt with Chain-of-Thought (CoT) reasoning

4.4. Fine-tuning

One notable characteristic of real-time control tasks is the need for rapid responses. OpenAI’s ChatGPT completion API offers fine-tuning¹⁰ for several advantages, including improved result quality, increased training capacity, token savings, and reduced latency. We concluded that minimizing latency is critical in an agentic environment, as our agent may be unable to correct piloting errors due to the dynamics of orbital mechanics, making precise adjustments challenging.

We also addressed the scarcity of training data by developing a script to record human gameplay in KSP and convert it into sequential data for training a model to imitate human-like spacecraft maneuvering. The fine-tuning dataset consists of observations and thrust actions recorded every 0.5 seconds during gameplay. Thrust actions are logged as discrete labels relative to the spacecraft’s reference frame (e.g., ‘backward,’ ‘forward,’ ‘left,’ ‘right,’ ‘down,’ ‘top’). For instance, the term ‘forward’ is mapped to a value of 1 in the forward thrust parameter, indicating maximum thrust.

To evaluate the performance of fine-tuning we run the following experiments sequentially, using *GPT-3.5-turbo-1106*¹¹ as LLM baseline:

- *Baseline LLM*: Standard “gpt-3.5-turbo” model with a generalized system prompt for Pursuer-Evader (PE) scenarios in the KSPDG challenge.
- *Fine-tuning*: Trained in a single human gameplay log (314 user-assistant pairs) with default hyperparameters.
- *+ Hyperparameter tuning*: Adjusting GPT-3.5 fine-tuning hyperparameters OpenAI (2023) improved performance. Lowering the learning rate multiplier to 0.2 enhanced exploration and convergence.
- *+ System prompt*: Including a refined system prompt in training to balance clear instructions without over-constraining model behavior.
- *+ Two train gameplays*: Adding another gameplay log (total 647 user-assistant pairs) to analyze learning improvements with more data.

GPT-4 and GPT-4o were unavailable during this study and are expected to perform better, as demonstrated in a related study Carrasco et al. (2025).

¹⁰GPT-3.5 Turbo fine-tuning and API updates: <https://openai.com/blog/gpt-3-5-turbo-fine-tuning-and-api-updates>

¹¹As of February 2024, following the conclusion of the KSPDG challenge, alternative models available with fine-tuning capabilities include *gpt-3.5-turbo-0613*, *babbage-002*, *davinci-002*, and *gpt-4-0613*. However, these demonstrated inferior performance in comparison.

5. LLaMA as a Spacecraft Operator

Right after the end of the KSPDG Challenge, there was an interesting approach left to experiment with. While designing and developing the agents, the limited interaction with the OpenAI API and the high cost of it led us to find open-sourced LLMs that could be tweaked and worked with more data and more domain knowledge.

LLaMA is a family of *large language models* released by Meta as a collection of foundational language models. The Hugging Face community, which also offers the *transformers* library Wolf et al. (2019), provides a platform for hosting an AI community where a plethora of different LLM artifacts, such as open-source models and specific datasets, can be found. For our work, we utilized LLaMA-3-8B, the smallest model in the LLaMA 3 family, and their instruct version, which is post-trained for chat interactions.

In these LLaMA experiments we were determined to solve some of the problems that arose during GPT experiments. It is obvious that the final result of the model may badly generalize, even when including Chain of Thought (CoT) which clearly decreases this problem although not perfect. The goal of our LLaMA research is to build on the promising fine-tuning results observed in GPT, aiming for improved performance through the development of a more adaptable and robust model.

Given the additional data, the discrete labels relative to the spacecraft’s reference frame included a new label for no action, called “none”, which will be useful for situations when no action is better than an action. The none action is situational and was very difficult to learn with limited data files in GPT. The reference frame for this LLaMA fine-tuning will be the same as GPT’s.

5.1. LLaMA Data

In an LLM, the tokenizer is typically unique to each model. This study leverages LLaMA 3, an open source model released in 2023 Touvron et al. (2023), to develop an autonomous pilot agent in Kerbal Space Program (KSP). The LLaMA 3 tokenizer includes special tokens (e.g., <|begin_of_text|><|end_of_text|>) that we used. For training, all files were converted to Alpaca CRFM (2023).

Listing 1: JSON Snippet in Alpaca Format.

```
{  
    "instruction": "\nHUMAN: Given these observations ...",  
    "output": "The best throttle is ...",  
    "system": "You operate as an autonomous agent ...",  
    "history": []  
}
```

Our LLaMA datasets utilized for training were entirely in Alpaca format, comprising ~ 100 gameplays.

Additionally, we extracted flight logs from Navball bot, a prograde-alignment system designed to minimize thrust and corrections, and converted them into a synthetic dataset with dynamic chain-of-thought prompts for training all LLaMA models. For more information about this bot see appendix Appendix E.

5.2. Few-shot Prompting

Before fine-tuning, a few-shot prompting with CoT was designed to test the baseline LLaMA 3 model capabilities. The prompt can be seen in Appendix C.

5.3. Optimization techniques

The AI research may be well bottlenecked due to the requirements needed to train any state-of-the-art model. During our LLaMA research, we utilized a workstation equipped with a Threadripper CPU¹² and five RTX 4090 GPUs¹³.

To maximize efficiency and improve the training process, we employed a range of optimization techniques and tools:

- **LoRA & DORA:** Reduce the number of trainable parameters by factorizing weight updates into low-rank and orthogonal low-rank matrices, enhancing efficiency and optimization Hu et al. (2024), Liu et al. (2024).
- **Quantization:** Initially applied to reduce model size and inference time, but was discarded due to performance degradation Nagel et al. (2021).
- **LLaMA Factory:** LLaMA-Factory Zheng et al. (2024) is a project that serves as a LLaMA framework which provides a comprehensive set of tools and scripts for fine-tuning, exporting, serving and benchmarking LLaMA models.
- **Flash Attention (FA) and FA2:** Optimize memory and computation in attention mechanisms, reducing training and inference latency Dao et al. (2022), Dao (2023).

¹²<https://www.amd.com/en/products/processors/workstations/ryzen-threadripper.html>

¹³<https://www.nvidia.com/es-es/geforce/graphics-cards/40-series/rtx-4090/>

5.4. Fine-tuning

To evaluate the performance of fine-tuning, we conducted the following sequential experiments using LLaMA-3 as the LLM baseline:

- Baseline LLaMA: This is the standard *LLaMA-3* model used without any fine-tuning. It is programmed with a generalized system prompt designed for the Pursuer-Evader (PE) scenarios within the KSPDG challenge. This prompt includes the key aspects of rendezvous missions.
- Fine-tuning 10 documents: Fine-tuning LLaMA with 10 navball agent logs. For testing the LLaMA capabilities with a small dataset.
- Fine-tuning 10 documents with sliding window of 3: Fine-tuning LLaMA with 10 navball agent logs using a sliding window of 3 actions. This model attempts to predict the next three actions based on the previous ones.
- Fine-tuning with 10 documents and look ahead: Fine-tuning LLaMA with 10 Navball agent logs using a look-ahead approach, where the agent outputs a series of sequential actions. This model is trained to predict the next three actions.
- Fine-tuning 25 documents: After increasing the training data to 25 navball agent logs. To test how LLaMA may improve given a medium dataset.
- Fine-tuning 50 documents: After further increasing the training data to 50 navball agent runs. To test the LLaMA fine-tuning with a rich set of gameplays.

Having outlined the iterations of each experiment, we now detail the hyperparameters used for fine-tuning:

Hyperparameter	Value
AdamW Optimizer	Yes
Quantization	None
Learning Rate	1e-4
Learning Rate Scheduler	Cosine
LoRA Configuration	$\{r = 16, \alpha = 8, d = 0.05\}$
Flash Attention 2	Enabled
Epochs	3
Dora	Enabled
Trainable Layers	Last 2 layers
Batch Size	2
Gradient Accumulation Steps	2

Table 1: Hyperparameters used for fine-tuning LLaMA

Gradient accumulation steps help compensate for the small batch size of 2, chosen due to GPU limitations and to balance convergence speed with generalization. LoRA parameters are optimized for stability, with only the top two layers trainable. Flash Attention 2 improves efficiency, while AdamW and a cosine learning rate scheduler facilitate smooth convergence. Dora dynamically adjusts the training parameters to further improve optimization.

6. Results

In this section, we evaluate the performance of various models. We will also use the metrics designed for the KSPDG Challenge Allen (2023) as well as some of our own metrics and evaluations.

6.1. Training Metrics

The accuracy of the inferred actions is assessed using a discrete evaluation metric that determines whether the predicted action matches the ground truth. Specifically, the overall action accuracy is defined as

$$\text{Acc}_{\text{action}}(\mathbf{a}, \hat{\mathbf{a}}) = \begin{cases} 1, & \text{if } \mathbf{a} = \hat{\mathbf{a}}, \\ 0, & \text{otherwise,} \end{cases} \quad (6)$$

where \mathbf{a} is the ground truth action vector and $\hat{\mathbf{a}}$ is the inferred action. Consequently, we compute a custom cross-entropy loss using the logistic loss function,

assuming that the LLM agent produces a one-hot encoded prediction corresponding to \hat{a} .

Next, the performance is quantified using KSPDG’s scoring function defined by

$$\begin{aligned} \text{Score} = & (0.1 \cdot d)^{2.0} + (0.5 \cdot v)^{1.5} \\ & + (0.1 \cdot f)^{1.25} + (0.01 \cdot t)^{1.0} \end{aligned} \quad (7)$$

where d , v , f , and t denote distance (m), velocity (m/s) fuel usage (l) and time (s), respectively. In each term, the component is scaled by a factor (e.g., 0.1, 0.5, etc.) and raised to an exponent (e.g., 2.0, 1.5, etc.). More generally, the scoring function is given by

$$\text{Score} = \sum_c (s_c \cdot c)^{w_c}, \quad (8)$$

with s_c and w_c representing the scale and exponent for component c , respectively.

6.2. Access to Experiments and Models

All experiments carried out during this investigation have been tracked using Weights & Biases (wandb). The projects are publicly accessible and provide detailed logs, metrics, and visualizations of the training processes. Additionally, the models developed and used in this research are hosted on Hugging Face. Each repository contains specific models and datasets relevant to different aspects of the research.

Details of the specific projects, including experimental runs, fine-tuning processes, and final versions of the models, including various versions of LLaMA models, can be found in the appendix. (See Appendix A and Appendix B)

6.3. GPT Training Results

Our GPT fine-tuning approach involved small datasets and a few runs due to the limitations that the API offers given its proprietary nature. Nonetheless, a minor fine-tuning was conducted using a small dataset to validate how well GPT adapts to the data and to examine the hypothesis that increased data ingestion leads to improved performance.

One key hyperparameter to highlight is the Learning Rate Multiplier (LRM). While OpenAI’s default settings are generally effective, the default LRM value of two proved problematic, causing rapid convergence and high overfitting. As summarized in table 2, our incremental experiments with GPT showed improvements in training loss but failed to yield similarly favorable validation loss outcomes. We attribute this discrepancy to the limitations of fine-tuning a non-locally hosted, closed-source

model, which restricts flexibility. This constraint ultimately motivated our shift toward open-source LLMs, allowing for greater control over the training process.

Model Name	Training Loss	Validation Loss
Final Models		
Default LRM	4.8×10^{-1}	-
+ LRM 0.2	2.3×10^{-2}	2.8×10^{-2}
+ system prompt	1.9×10^{-1}	5.2×10^{-3}
+ 2 human gameplays	1.3×10^{-4}	4.4×10^{-1}

Table 2: Training and Validation Loss for Each GPT Model

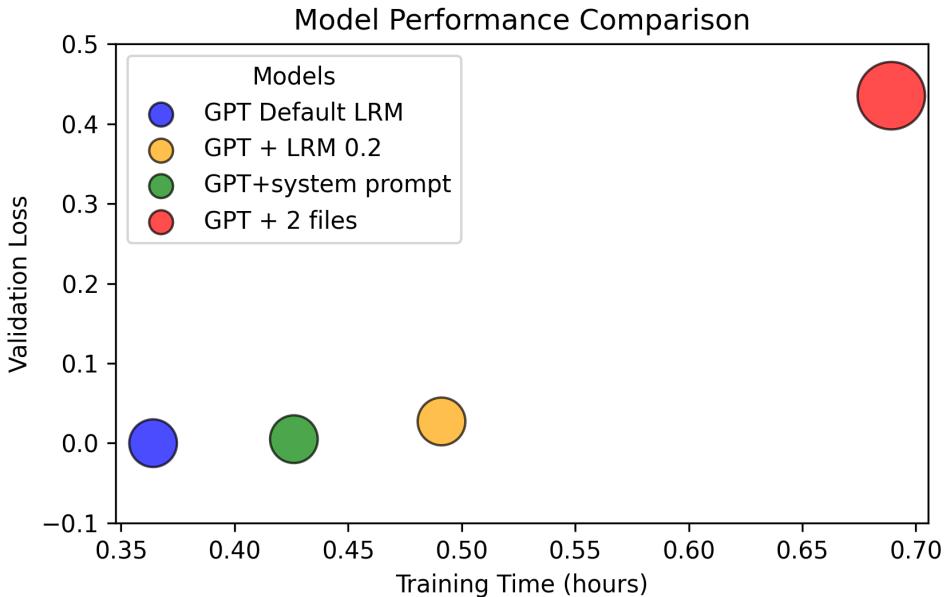


Figure 5: Model Performance with Final Hyperparameters (GPT). Circle size represents training dataset size. GPT Default LRM lacked a validation set, marked as zero.

6.4. LLaMA Training Results

Our fine-tuned LLaMA models can be categorized into two distinct groups: the experimental models, which were trained during a phase where we experimented with

variations of hyperparameters, and the final models, which incorporate all the final hyperparameters presented in table 1.

When comparing the experimental models to the final versions, we observed a significant improvement on the order of one to two magnitudes. Table 3 shows the losses for all models. As also shown in fig. 6, models with larger datasets generally achieve better results. However, an exception is the model that uses a sliding window of 3, which, despite having fewer files, performs similarly to the 50-file model. This indicates that the sliding window approach is highly effective in enabling the model to learn from fewer gameplays. There was an experiment attempting to predict the next ‘x’ steps of the spacecraft, referred to as the ‘look ahead’ (la) method. However, it was not very effective and is therefore symbolically explained in the methodology.

Model Name	Training Loss	Validation Loss
Experimental Models		
Huggingface Models	6.8×10^{-1}	7.0×10^{-1}
LLaMA-Factory Models	2.8×10^{-2}	3.7×10^{-2}
Final Models		
LLaMA 35 files quant	1.5×10^{-2}	1.4×10^{-3}
LLaMA 10 files la	5.7×10^{-2}	4.5×10^{-2}
LLaMA 10 files win	1.1×10^{-2}	4.8×10^{-3}
LLaMA 10 files	2.8×10^{-2}	2.2×10^{-2}
LLaMA 25 files	1.6×10^{-2}	5.7×10^{-3}
LLaMA 50 files	1.0×10^{-2}	4.2×10^{-3}

Table 3: Training and Validation Loss for Each LLaMA Model

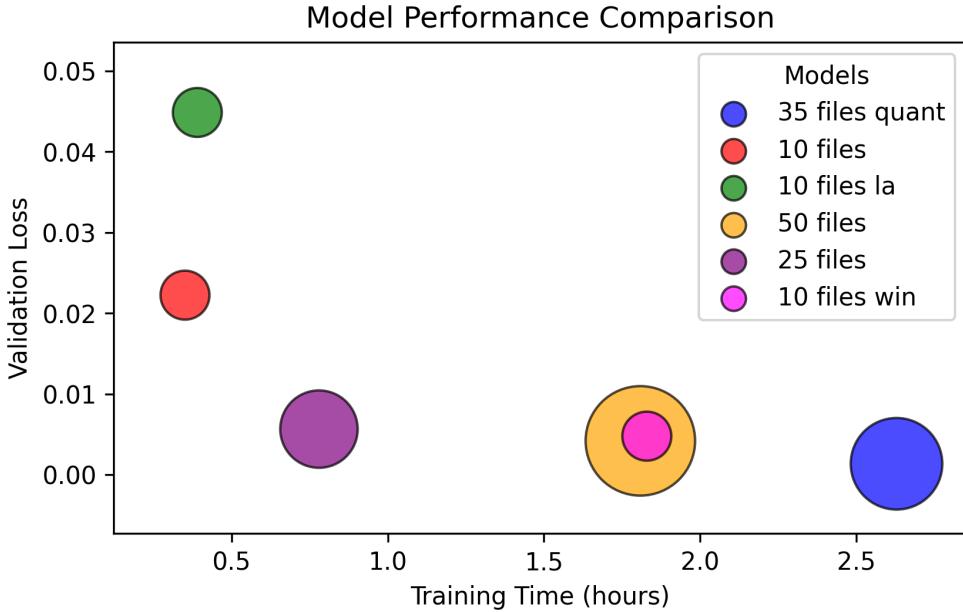


Figure 6: Comparison of Model Performance with Final Hyperparameters and Various Dataset Sizes (LLaMA). “Quant” indicates models with quantized parameters, “la” refers to look ahead, and “win” denotes models trained with a sliding window dataset approach. The size of the circle refers to the size of the dataset.

Throughout the chronology of all the different training runs we can see how the model is enhanced due to the gradual improvements, which demonstrates the effectiveness of the path chosen for these trainings.

All evaluations were conducted using LLaMA’s 8B Instruct model, which was selected based on GPU memory capacity, inference speed considerations, and its foundational knowledge and format compatibility.

6.5. GPT Performance

Interestingly, while the OpenAI API for fine-tuning requires customization, it provides limited tools, especially compared to LLaMA. Hence, the effectiveness of GPT training is heavily dependent on the quantity and quality of data, together with certain adjustments (notably hyperparameters), with the LRM having the most significant impact.

The application of the Chain of Thought approach demonstrates a significant improvement in the generalization of spacecraft piloting techniques for the Pursuer-Evader problem as well as guiding the model to achieve a 0% failure rate in execution.

Results in Table 4 indicate a consistent achievement of target proximity within 25 meters across varied scenarios, highlighting the approach’s effectiveness in enhancing model precision and adaptability in complex rendezvous tasks. It is evident that this advancement notably enhances the model reasoning for its spacecraft operating capabilities.

Method	E1 (m)		E2 (m)		E3 (m)		E4 (m)		Failure Rate
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	
Naive	229.75	267.51	182.28	236.10	225.0	225.0	215.20	216.49	-
Baseline GPT	36.42	269.22	245.76	295.09	292.59	328.20	305.02	329.78	37.5%
w/ CoT	11.02	14.59	15.73	21.28	5.63	21.71	10.42	17.27	0.00%

Table 4: Performance of the GPT agent (closest distance) with Chain of Thought (CoT) prompting across different Pursuer-Evader environments.

Table 5 summarizes the performance of our fine-tuned agents. An error in calling a function causes the environment to terminate or the agent to freeze; these are considered failures. During all successful runs, we calculate the best and average closest approach distance. We compare these results against the agents used in SpaceGym Allen et al. (2023). Specifically, we compare them against a naive agent, a Lambert model predictive control (Lambert-MPC) agent, an iLQGames agent Fridovich-Keil et al. (2020), and a Proximal Policy Optimization (PPO) agent, as benchmarked in the SpaceGym paper Allen et al. (2023). We refer the reader to SpaceGym to read more about these agents.

As shown in the table, each fine-tuning experiment improves our agent’s performance while also reducing run variance. Notably, fine-tuning significantly decreases GPT-3.5 response latency, as demonstrated in Table 4. This improvement is primarily due to the reduced number of output tokens generated by the fine-tuned models, as well as the high failure rate observed in the GPT baseline (Table 4), where most failures stem from function call invocation errors.

Additionally, a key insight from both Table 5 and Table 6 is that fine-tuning yields progressively better results with continued refinements. This suggests that our fine-tuning strategy has yet to reach a performance plateau, indicating that incorporating more data could further enhance our agents’ capabilities and unlock even greater improvements.

As shown in Table 4, the GPT-3.5 baseline model’s failure rate in KSP scenarios exceeds tolerable limits, while fine-tuned models reduce failures to nearly 0%. Crucially, only with all incremental refinements do we surpass the baseline prompt-engineered model without chain of thought, underscoring the need for more training samples.

Method	Best Dist. (m)	Avg Dist. (m)	Failure Rate
Naive	225.0	225.0	-
PPO	>1643	2346	-
iLQGames	>53.31	60.99	-
Lambert-MPC	>18.24	47.94	-
Baseline GPT	178.11	200.10	36.8%
Simple fine-tuning	263.55	265.89	0.0%
+ Hyperparam tuning	188.90	202.08	0.1%
+ System prompt	197.41	214.87	0.0%
+ Two train gameplays	132.09	159.78	0.2%
Human Gameplay	5.97	6.25	-

Table 5: Performance of GPT for each fine-tuning technique in meters.

Method	Best Latency	Average Latency	Standard Deviation
baseline GPT	759.69	840.42	83.85
simple fine-tuning	977.32	987.43	15.08
+ hyperparameter tuning	749.02	831.30	49.02
+ system prompt	684.13	753.52	51.75
+ two train gameplays	468.98	557.49	89.54

Table 6: Latency of GPT responses for each experiment in milliseconds.

As illustrated in fig. D.11, there is a notable issue when comparing the current model with the GPT agent, specifically regarding overshooting. This problem arises from an incorrect prompt hint (see section 4.4) to *accelerate in the direction of the relative position of the evader*. Consequently, this forces the model to adopt an accelerate-only approach. This issue will be addressed in the subsequent LLaMA experiments.

6.6. LLaMA Performance

In contrast to the GPT experiments, the LLaMA experiments were specifically designed to achieve superior fine-tuning results. This focus is due to the inherent nature of an open-source model, that is intended to be enhanced and customized by its users.

The iterative process for all experiments is explained in section 5.4. Table 1 summarizes the performance metrics of the LLaMA fine-tuning experiments.

Method	Best Distance (m)	Average Distance (m)	Failure Rate
Naive	225.0	225.0	-
PPO	>1643	2346	-
iLQGames	>53.31	60.99	-
Lambert-MPC	>18.24	47.94	-
baseline LLaMA	52.69	140.68	9.09%
fine-tune 10 files	30.52	51.53	0.00%
fine-tune 25 files	13.54	29.44	0.00%
fine-tune 50 files	11.86	29.76	0.00%
fine-tune 10 files win=3	23.08	40.03	0.00%
Navball (bot)	34.34	36.43	-

Table 7: Performance of LLaMA for each fine-tuning technique in meters. The scenario used for these results is E3.

The LLaMA results have exceeded our expectations. Not only did the model follow a robust prograde trajectory (see section 6.8) but it also outperformed almost every other method in the KSPDG Challenge. While these results are second only to the Chain of Thought (CoT) approach, it could be argued that CoT “hacks” the language model via prompting, potentially undermining its intrinsic reasoning capabilities. It is noteworthy that all dataset files were collected using the navball bot, and the LLaMA fine-tuned model moderately outperforms the navball bot’s performance.

It is important to discuss that the base LLaMA model achieves better results than the GPT model. However, this outcome is expected when considering that LLaMA-3 competes with GPT-4, rather than GPT-3 which is the model that was used in our GPT approach.

One technique utilized in these experiments but not in the GPT experiments due to limitations was the sliding window method (see section 4.1). As evidenced by the training and experimental results, despite having a significantly smaller data set, the sliding-window model competes with the top three best performing fine-tuned models.

Furthermore, we can see the latency results in table 8. The observed latencies indicate a significant delay that adversely affects the model’s performance when utilized by an agent, especially when compared to our GPT models. Optimizing the model or upgrading the technical resources may lead to better results.

Ultimately, as illustrated in the appendix in fig. D.10, the best-performing models exhibit a “clunky” movement pattern due to action delays. However, the overall distributions are quite similar. The navball bot should serve as the benchmark to evaluate the models’ performance.

Method	Best Latency	Average Latency	Standard Deviation
baseline LLaMA	6449.16	8580.43	1175.69
LLaMA ft 10 files	3305.15	3444.88	21.28
LLaMA ft 25 files	3282.21	3316.89	21.19
LLaMA ft 50 files	3418.97	3455.29	31.22
LLaMA ft 10 files win=3	3252.89	3292.44	38.93

Table 8: Latency of LLaMA responses for each experiment in milliseconds.

6.7. Scoring and Accuracy results

Table 9 presents the loss and accuracy of the cross-entropy of various fine-tuned Llama and GPT models in scenario E3. Notably, the LLaMA models demonstrate excellent results, significantly outperforming the GPT models.

Model	Accuracy	Cross-Entropy Loss
Llama 10 files	0.83	5.95
Llama 25 files	0.80	7.27
Llama 50 files	0.96	1.32
Llama 10 files win=3	0.95	1.65
GPT Default LRM	0.049	34.27
GPT + LRM 0.2	0.20	28.76
GPT + system prompt	0.37	22.55
GPT + 2 human gameplays	0.26	26.69

Table 9: Cross-entropy loss and Accuracy of Llama and GPT Fine-tuned Models in Scenario E3. These metrics were only used on trained models.

The results shown in table 9 indicate that the Llama fine-tuning yields exceptional performance, with significantly higher accuracy and cross-entropy loss compared to the GPT models. It's important to note that the GPT models only had access to 2 files, which likely limited their performance.

This performance extends beyond the training distribution, highlighting a remarkable outcome: the LLaMA models outperformed the navball bot that originally generated the training data. This unexpected result suggests that the LLaMA models were able to leverage their prior knowledge and reasoning capabilities to make more effective decisions, surpassing the baseline set by the navball bot. The discrepancy in accuracy underscores the potential of fine-tuned models to generalize beyond their training data, demonstrating a level of adaptability that was not initially anticipated.

The fine-tuned LlaMA Model with 50 files, achieved the highest accuracy and the lowest cross-entropy, closely followed by the 10 files using sliding window model.

This performance underscores the model’s ability to generalize better from a larger dataset, significantly optimizing its decision-making process.

The results of the KSPDG scoring evaluation are presented in table 10 below. Please note that best speed means best **approach** speed, this means the speed of the pursuer at the closest distance.

Experiment	Best Score	Best Distance	Best Speed	Best Fuel Usage
GPT Experiments				
GPT Baseline	759.69	178.11	77.77	651.54
GPT Default LRM	977.32	263.55	74.62	<u>220.84</u>
GPT + LRM 0.2	749.02	188.90	72.66	611.55
GPT + system prompt	684.13	197.41	75.91	<u>235.49</u>
GPT + 2 files	468.98	132.09	73.58	<u>267.73</u>
LLaMA Experiments				
LLaMA Base Model	387.55	140.68	50.76	390.77
LLaMA 10 files (optimized)	232.96	30.52	19.26	615.11
LLaMA 25 files (optimized)	<u>184.31</u>	<u>13.54</u>	<u>18.01</u>	436.02
LLaMA 50 files (optimized)	194.71	<u>11.86</u>	27.29	471.58
LLaMA 10 files win=3 (optimized)	212.17	23.08	24.83	462.08
Training Data Distribution				
Navball (bot)	63.64	34.34	<u>16.62</u>	123.19
Human gameplay	<u>72.39</u>	5.97	3.77	1125.70

Table 10: Best Parameters and Scoring for Different Models. **Bold** indicates best, underline indicates second best, and dashed underline indicates third best.

Since the LLaMA fine-tuning took place after the conclusion of the KSPDG Challenge, scoring was not the primary focus of our approach. Instead, we prioritized optimizing the model to minimize the best distance, which is reflected in the results.

Additionally, the model’s latency increases the likelihood of failing minor adjustments and prograde alignments, resulting in higher propellant consumption for corrections. Even so, the results for the other factors are far from bad.

For the GPT runs, although human gameplay is superior, we observe gradual improvements and better fuel efficiency across iterations. However, this efficiency arises because the model often “decides” not to move.

6.8. Comparison of trajectories

There were some worth noticing patterns that helped us improve the models and validate their reasoning during the KSP gameplays.

The figures in fig. 7b and fig. 7a illustrate key points discussed earlier. The red line represents the trajectory of the evader.

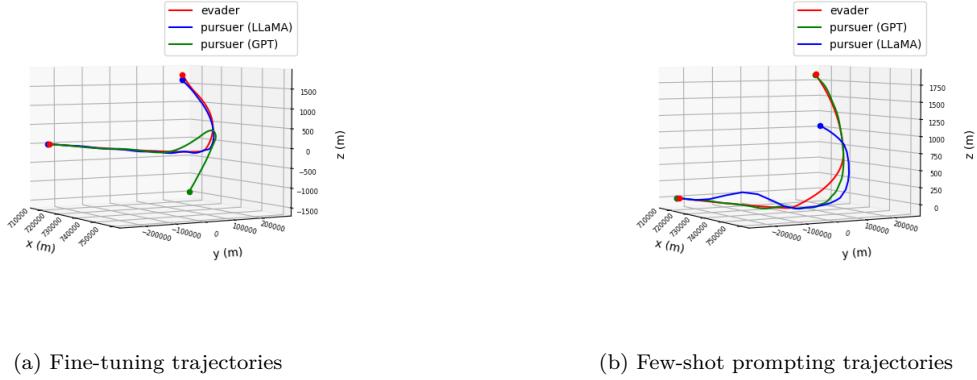


Figure 7: Comparison of fine-tuning and few-shot prompting trajectories. The red line represents the trajectory of the evader.

The fine-tuning trajectories in fig. 7a indicate that the data ingested by the model aids in understanding the problem and determining appropriate actions. However, these trajectories also show that the model’s prior knowledge and reasoning still influence its performance. For instance, an incorrect hint, as depicted by the GPT trajectory (green line), deteriorates the model’s performance and makes the agent ‘overshoot’ (receding from the evader). In contrast, an “agnostic” prompt that does not dictate the model’s reasoning can even enhance data-driven performance.

On the other hand, the few-shot prompting trajectories using the Chain of Thought technique are similar between GPT and LLaMA models. GPT performed slightly better due to its significantly lower latency.

7. Conclusions & Future Work

We find the results very satisfying. The spacegym integration, alongside the orbit generation and agent implementation, demonstrated that Kerbal Space Program can be a great, yet simple, alternative as a simulation engine.

The findings obtained within a brief training period of two to four months for model development and agent implementation provide strong evidence of the effectiveness of this approach as a robust and viable alternative. We successfully created an agent that could mimic and even improve upon “expert” data such as synthetically generated data or human gameplays.

Following this, there is no doubt that training a LLM can leverage prior knowledge and improve it for specific scenarios. The LLaMA agent, as an open-sourced model, was more complex to train but offered a plethora of tools, frameworks, and

hyperparameters. These LLaMA models significantly outperformed the GPT ones, which achieved second place in the KSPDG Challenge, only behind a differential equations approach, and not too far from this said approach.

It is important to note that the base LLaMA model achieves better results than the GPT model. However, this outcome is expected when considering that the LLaMA-3 model competes with the GPT-4 model, rather than the GPT-3 model used in the original GPT approach.

Enhancing the model’s optimization or upgrading the technical resources may lead to improved results, as these outcomes are highly hardware-dependent. In contrast, the GPT models were fully trained using OpenAI’s powerful resources, which completely outweigh our local ones. Although the LLaMA model’s results are already better, it is important to note that there is still room for further improvement.

Moreover, it is crucial to reflect on and compare these results with what would be obtained if reinforcement learning (RL) were used. The largest tests conducted here, which yielded very good results, involved barely 50 files. In contrast, RL would typically require thousands of examples or games to achieve comparable performance. This highlights the efficiency of LLMs in learning from a relatively small dataset, whereas RL approaches would need a significantly larger amount of data to train effectively. In future work, while our results indicate that RL is not a suitable alternative in this specific scenario, we aim to explore the potential benefits of integrating both approaches to enhance overall performance. However, it is important to note that the simulation environment used in this study is designed for test-time computation and is not conducive to on-policy learning methods.

8. Acknowledgements

This research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. This work has also been supported by the MISTI-UPM program and by the Madrid Government (Comunidad de Madrid-Spain) under the Multianual Agreement 2023-2026 with Universidad Politécnica de Madrid in the Line A, “Emerging PhD researchers”. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. Authors would like to thank Dr. Ross Allen and the rest of the team behind the development of the KSPDG challenge for the technical support and the creation of the competition.

Appendix A. Detailed Weights & Biases Projects

- **LLaMA Experimental:**
 - **Description:** This project contains the experimental runs and fine-tuning processes for the LLaMA models.
 - **Link:** [LLaMA Experimental Project](#)
- **LLaMA Final:**
 - **Description:** This project includes the final versions of the LLaMA models with optimized parameters and performance metrics.
 - **Link:** [LLaMA Final Project](#)
- **GPT Project:**
 - **Description:** This project documents the experiments and results for the GPT models used in the KSPDG Challenge.
 - **Link:** [GPT Project](#)

Appendix B. Detailed Hugging Face Models

- **LLaMA 35 files quantized:**
 - **Link:** [LLaMA 35 files quantized](#)
- **LLaMA 10 files:**
 - **Link:** [LLaMA 10 files](#)
- **LLaMA 10 files sliding window:**
 - **Link:** [LLaMA 10 files sliding window](#)
- **LLaMA 10 files look ahead:**
 - **Link:** [LLaMA 10 files look ahead](#)
- **LLaMA 25 files:**
 - **Link:** [LLaMA 25 files](#)
- **LLaMA 50 files:**
 - **Link:** [LLaMA 50 files](#)

Appendix C. Prompting

You operate as an autonomous agent controlling a pursuit spacecraft. Your goal is to apply throttles to capture the evader given the positions and velocities of the pursuer and evader in celestial body reference frame and the direction of pursuer's velocity relative to evader or prograde. Throttles must be given in your vessel reference frame wherein the x axis points to the right, the y axis points towards the target and the z axis points upwards. The maximum throttle is 1. Reason step-by-step.

Figure C.8: System prompt used both for the few-shot prompting and fine-tuning in LLaMA.

HUMAN: Given these observations ... , what is the best throttle to capture evader?
ASSISTANT: The x coordinate of prograde is positive, indicating that pursuer is moving to the right. The y coordinate of prograde is negative, indicating that evader is approaching. The z coordinate of prograde is negative, indicating that pursuer is moving down. **To capture the evader we should counteract pursuer's motion, moving in the opposite direction in the x axis, towards the target in the y axis, and in the opposite direction in the z axis.** This means we should apply throttles to move left, forward and up. Therefore we should call `perform_action(ft: forward, rt: left, dt: up)`.
Now answer the following question: Given these observations ... , what is the best throttle to capture evader?. **Reason step-by-step.**

Figure C.9: User prompt used both for the few-shot prompting and fine-tuning in LLaMA.

Appendix D. Trajectories

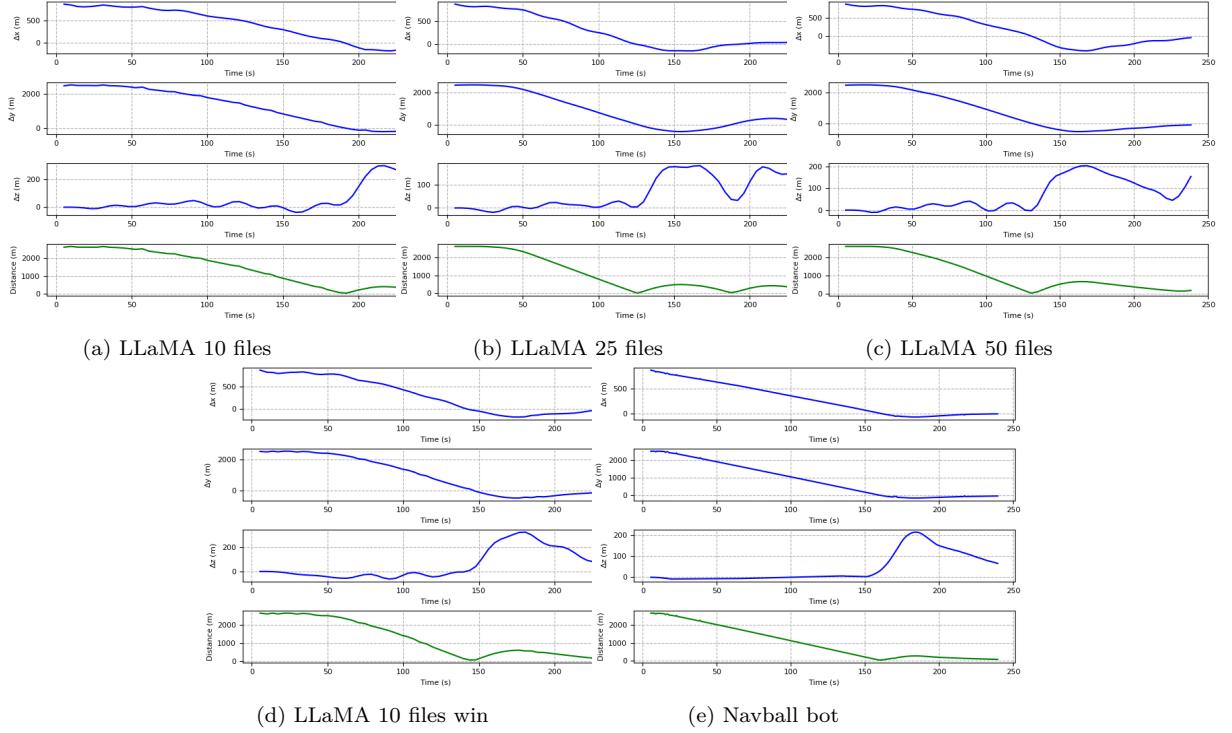
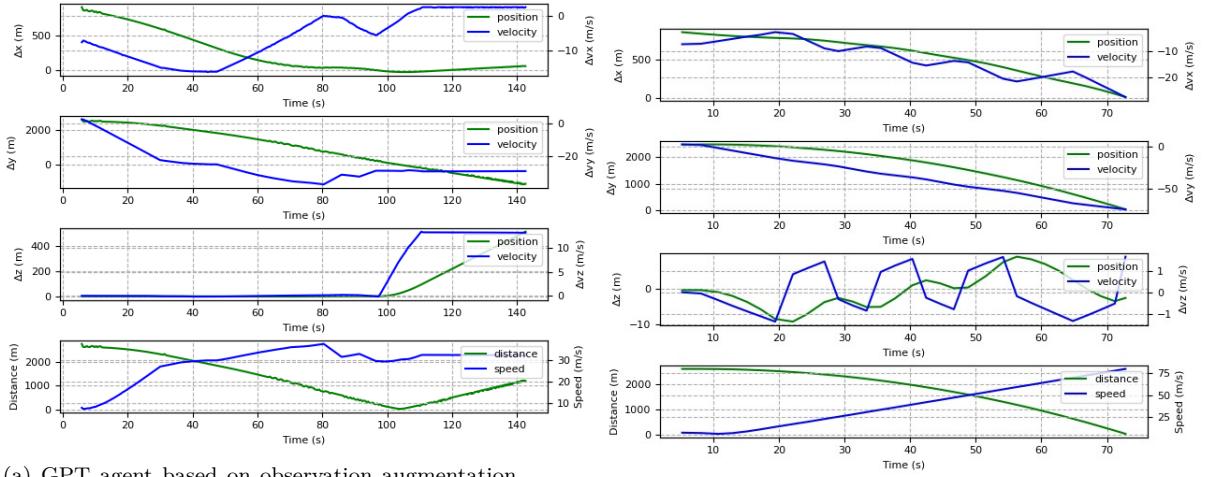


Figure D.10: Behavior of the developed LLaMA models in Pursuer-Evader scenario E3. The plots show the evolution of differences in position, velocity, as well as the relative distance and velocity metrics.



(a) GPT agent based on observation augmentation and few-shot prompting (excluding CoT).

(b) GPT agent based on fine-tuning.

Figure D.11: Behavior of the developed GPT agents in Pursuer-Evader scenario E3. The plot shows the evolution of differences in the position, velocity, as well as the relative distance and velocity metrics.

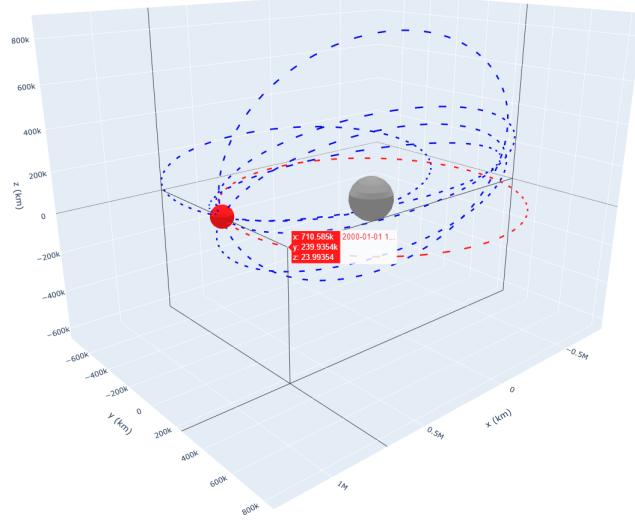


Figure D.12: An example of the generation of multiple orbits used to collect LLaMA training data

Appendix E. Algorithms

Algorithm 1 LLM Agent Pseudocode

```
1: Define class LLMAgent(KSPDGBaseAgent):
2:   Initialize with scenario settings, prompt templates, and agent's configuration
      parameters like sliding window size
3:
4:   Define get_action(observation)
5:     Build new state with observations and augmented data (e.g. prograde)
6:     Generate prompts (system / user) for new state
7:     Add conversation history if sliding window is used
8:     Try:
9:       Call LLM API
10:      If response includes function call Then
11:        Extract action from function call
12:      Else If action can be extracted from response Then
13:        Extract action from the response
14:      Else
15:        Raise FailureException
16:      Append (state, action) pair to sliding window
17:      Log LLM interaction: state, action, input and output prompts
18:      Return action
19:    Except (FailureException):
20:      Log the failure
21:      Wait for one second
22:      Return None or a default action
23:
24: Main program:
25:   Set scenario and environment
26:   Create instance of LLM Agent
27:   Create and run the environment's agent
```

Algorithm 2 Navball Agent Algorithm

- 1: **Define** get_action(observation):
- 2: Calculate prograde
- 3: **if** the deviation of the vessel's alignment from the prograde vector in the left-right or down-up directions exceeds ROTATION_THRESHOLD **then**
- 4: Apply rotational throttles to center the prograde marker on the navball
- 5: **else**
- 6: Do not apply rotation throttles
- 7: **end if**
- 8: Calculate the braking distance if maximum backward throttle is applied, assuming uniform deceleration:

$$d = \frac{1}{2} \frac{v_0^2}{a} \quad (\text{E.1})$$

where v_0 is the initial relative speed and a is VESSEL_ACCELERATION.

- 9: **if** the vessels are moving apart **or** the braking distance is less than the current distance **then**
- 10: Apply forward throttle
- 11: **else**
- 12: **if** the approach speed exceeds APPROACH_SPEED **then**
- 13: Apply backward throttle
- 14: **else**
- 15: Do not apply thrust in the forward direction
- 16: **end if**
- 17: **end if**
- 18: **return** action

Algorithm 3 Orbit Generation Pseudocode

- 1: **repeat**
- 2: Randomly generate eccentricity, inclination, and initial distance within given constraints
- 3: Estimate the radial distance as a fraction of the initial distance
- 4: Calculate the semi-major axis considering that the radius is evader's orbit radius and correct it by adding the estimated radial distance. Where:

$$r = \frac{a(1 - e^2)}{1 + e \cos(\nu)} \quad (\text{E.2})$$

$$v = \sqrt{\mu \left(\frac{2}{r} - \frac{1}{a} \right)} \quad (\text{E.3})$$

- 5: Set argument of periapsis and longitude of the ascending node to the evader's orbit argument of periapsis and true anomaly.
 - 6: Set true anomaly to zero
 - 7: Accept the generated orbit if it meets the distance constraints
 - 8: **until** the desired number of orbits are generated
-

References

- Allen, R., 2023. spacegym-kspdg. <https://github.com/mit-l11/spacegym-kspdg>. Accessed: October 4, 2023.
- Allen, R.E., Rachlin, Y., Ruprecht, J., Loughran, S., Varey, J., Viggh, H., 2023. Spacegym: Discrete and differential games in non-cooperative space operations, in: 2023 IEEE Aerospace Conference, pp. 1–12. doi:10.1109/AERO55745.2023.10115968.
- Anthropic, 2023. Claude: An ai assistant by anthropic. URL: <https://www.anthropic.com/index/clause>. accessed: 2024-07-03.
- Arslanian, H., Fischer, F., 2019. Understanding Artificial Intelligence and Its Capabilities. Palgrave Macmillan, Cham. chapter 14. pp. 45–64. doi:10.1007/978-3-030-14533-0_14.
- Beltagy, I., Peters, M.E., Cohan, A., 2020. Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150 .

- Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165 .
- Carrasco, A., Nedungadi, M., Rodriguez-Fernandez, V., Linares, R., 2025. Visual language models as operator agents in the space domain, in: AIAA SCITECH 2025 Forum, AIAA. p. N/A. URL: <https://arc.aiaa.org/doi/abs/10.2514/6.2025-1543>, doi:10.2514/6.2025-1543, arXiv:<https://arc.aiaa.org/doi/pdf/10.2514/6.2025-1543>.
- CRFM, S., 2023. Alpaca: An instruction-following llama model. <https://crfm.stanford.edu/2023/03/13/alpaca.html>. Accessed: 2024-07-15.
- Dao, T., 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. URL: <https://arxiv.org/abs/2307.08691>, arXiv:[arXiv:2307.08691](https://arxiv.org/abs/2307.08691). arXiv preprint arXiv:2307.08691.
- Dao, T., Fu, D.Y., Ermon, S., Rudra, A., Ré, C., 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. URL: <https://arxiv.org/abs/2205.14135>, arXiv:[2205.14135](https://arxiv.org/abs/2205.14135).
- DeepMind, 2023. Gemini: A general-purpose transformer model. URL: <https://www.deepmind.com/publications/gemini>. accessed: 2024-07-03.
- DeepSeek-AI, 2024. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437 arXiv:[2412.19437](https://arxiv.org/abs/2412.19437). available at <https://github.com/deepseek-ai/DeepSeek-V3>.
- Dettmers, T., Touvron, H., Joulin, A., Grave, E., Raileanu, R., Ott, M., 2023. Mistral 7b. URL: <https://arxiv.org/abs/2310.06825>, arXiv:[2310.06825](https://arxiv.org/abs/2310.06825).
- Engadget, 2014. Kerbal space program asteroid redirect mission add-on now available. URL: <https://www.engadget.com/2014-04-02-kerbal-space-program-asteroid-add-on.html>. accessed: 2024-07-02.
- Fridovich-Keil, D., Ratner, E., Peters, L., Dragan, A.D., Tomlin, C.J., 2020. Efficient iterative linear-quadratic approximations for nonlinear multi-player general-sum differential games, in: 2020 IEEE international conference on robotics and automation (ICRA), IEEE. pp. 1475–1481.

- Gaudet, B., Linares, R., Furfaro, R., 2020. Deep reinforcement learning for six degree-of-freedom planetary landing. *Advances in Space Research* 65, 1723–1741.
- Howard, J., Ruder, S., 2018. Universal language model fine-tuning for text classification, in: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 328–339. URL: <https://aclanthology.org/P18-1031/>.
- Hu, E.J., Shen, Y., Wallis, P., Yao, Z., Chen, W., Wang, H., Dong, Z., Shafran, I., 2024. Lora: Low-rank adaptation of large language models. URL: <https://arxiv.org/abs/2402.12354>, arXiv:2402.12354.
- Ji, Z., et al., 2023. Hallucinations in large language models: Prevalence, causes, and mitigation strategies. *arXiv preprint arXiv:2305.14390* URL: <https://arxiv.org/abs/2305.14390>.
- Kerbal Space Program Wiki, 2018. Navball. URL: <https://wiki. kerbalspaceprogram.com/wiki/Navball>. accessed: 2025-02-23.
- Lee, K., Firat, O., Agarwal, A., Fannjiang, C., Sussillo, D., 2018. Hallucinations in neural machine translation, in: *International Conference on Learning Representations (ICLR)*, p. N/A. URL: <https://openreview.net/forum?id=SkxJ-309FQ>.
- Liu, S.Y., Wang, C.Y., Yin, H., Molchanov, P., Wang, Y.C.F., Cheng, K.T., Chen, M.H., 2024. Dora: Weight-decomposed low-rank adaptation. URL: <https://arxiv.org/abs/2402.09353>, arXiv:2402.09353.
- Mao, J., Qian, Y., Zhao, H., Wang, Y., 2023. Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415* .
- Nagel, M., van Baalen, M., Blankevoort, T., Welling, M., 2021. White noise: Regularizing neural networks through quantization, in: *International Conference on Machine Learning*, PMLR. pp. 7197–7206.
- OpenAI, 2022. Introducing chatgpt. <https://openai.com/blog/chatgpt>. Accessed on 03/29/2024.
- OpenAI, 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* .
- OpenAI, 2023. Iterating on hyperparameters - fine-tuning guide. <https://platform.openai.com/docs/guides/fine-tuning/iterating-on-hyperparameters>. Accessed: 2024-03-28.

- OpenAI, 2024. Introducing openai o1-preview. URL: <https://openai.com/index/introducing-openai-o1-preview/>. accessed: 2025-02-06.
- Piaget, J., 1926. The Language and Thought of the Child. Harcourt, Brace & Company.
- Reynolds, L., McDonell, K., 2021. Prompt programming for large language models: Beyond the few-shot paradigm. URL: <https://arxiv.org/abs/2102.07350>. accessed: 2024-07-02.
- Rodriguez-Fernandez, V., Carrasco, A., Cheng, J., Scharf, E., Siew, P.M., Linares, R., 2024. Language models are spacecraft operators. arXiv preprint arXiv:2404.00413 URL: <https://arxiv.org/abs/2404.00413>.
- Russell, S., Norvig, P., 2016. Artificial Intelligence: A Modern Approach. Pearson Education Limited.
- Siew, P.M., Jang, D., Roberts, T.G., Linares, R., 2022. Space-based sensor tasking using deep reinforcement learning. *The Journal of the Astronautical Sciences* 69, 1855–1892.
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al., 2023. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 .
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need, in: Advances in Neural Information Processing Systems, p. N/A.
- Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al., 2023. A survey on large language model based autonomous agents. arXiv preprint arXiv:2308.11432 .
- Wei, J., et al., 2022. Chain of thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903 URL: <https://arxiv.org/abs/2201.11903>.
- Wittgenstein, L., 1953. Philosophical Investigations. Blackwell.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q.,

- Rush, A.M., 2019. Transformers: State-of-the-art natural language processing. URL: <https://arxiv.org/abs/1910.03771>, arXiv:1910.03771.
- Wu, Y., Min, S.Y., Prabhumoye, S., Bisk, Y., Salakhutdinov, R., Azaria, A., Mitchell, T., Li, Y., 2023. Spring: Gpt-4 out-performs rl algorithms by studying papers and reasoning. arXiv preprint arXiv:2305.15486 .
- Zheng, Y., Zhang, R., Zhang, J., Ye, Y., Luo, Z., Feng, Z., Ma, Y., 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), Association for Computational Linguistics, Bangkok, Thailand. p. N/A. URL: <http://arxiv.org/abs/2403.13372>.