Creating a Scalable Virtual Flight Instructor Using Large Language Models

Colin Sullivan, Kyle Tauzer
Lockheed Martin
Orlando, FL
colin.m.sullivan@lmco.com,
kyle.a.tauzer@lmco.com

Mark Cavanagh, Christopher Lee, Jean Seda
Lockheed Martin
Orlando, FL
mark.cavanagh@lmco.com,
christopher.1.lee@lmco.com, jean.seda@lmco.com

ABSTRACT

As the demand for skilled pilots continues to grow, the need for innovative approaches to pilot training has become increasingly urgent. Virtual flight instructors, when integrated into flight simulators, have promised the best of all worlds: tailored feedback adapted to the student, and self-paced access to learning materials without the need for human instructor oversight. Current systems provide detailed analysis by making use of state-based architectures. Development requires engineers and pilot subject matter experts to manually design, implement, and script each training scenario and the corresponding feedback. This manual approach creates excellent demo capabilities, but limits the scalability, flexibility, and affordability of these virtual instructors to meet full syllabi or address more complex scenarios.

To address these challenges, this paper experiments with leveraging open-source Large Language Models (LLMs) to create a scalable, simulator agnostic virtual instructor testbed. The prototype application ingests raw simulator data and generates detailed analysis and feedback of student pilot performance. To enable this capability, techniques such as agentic artificial intelligence (AI), chain prompting, and retrieval augmented generation (RAG) are experimented with. A RAG mechanism is prototyped that searches a comprehensive database of vectorized flight information including aircraft manuals, FAA documents, mission data, and other relevant flight data. This approach more closely mimics real-life instructors by focusing on embedding comprehensive understanding of the aircraft and mission sets into the virtual instructor, instead of relying on rote evaluation against defined rules. This paper reviews how various design tradeoffs made during development impacted the measures of reliability, quality of feedback generated, and scalability to other applications.

ABOUT THE AUTHORS

Colin Sullivan is the Principal Investigator for the described AI Pilot trainer. He received his Bachelor of Science in Computer Science from the University of Michigan in 2018. Colin has led the technical development of multiple training systems, including augmented reality platforms. His areas of research include artificial intelligence, augmented reality, and 3D modeling.

Kyle Tauzer is a Program Management Senior Manager at Lockheed Martin Rotary and Mission Systems (RMS). He holds a Bachelor of Science in Aerospace Engineering (BSAE) from the University of Michigan, and a Master of Business Administration (MBA) from the University of Florida. Kyle has 14 years of program management experience overseeing a variety of defense training and simulation programs. Currently, he serves as the program manager for all advanced technology programs and product lines for the RMS Training, Logistics, and Simulation business.

Mark Cavanagh is an AI Research Engineer at Lockheed Martin Rotary and Mission Systems (RMS). He holds a Bachelor of Science in Electrical Engineering (BSEE) and a Master of Science in Electrical Engineering (MSEE) from the University of South Florida. With a strong background in Artificial Intelligence and Machine Learning, Mark has applied his expertise across various domains, including biometrics and pilot training. Currently, he serves as the Principal Investigator for a portfolio of AI research and development projects within the RMS Training, Logistics, and Simulation business.

Christopher Lee is a software engineer on the AI Frontiers consulting team at Lockheed Martin Rotary and Mission Systems (RMS). He is an alumnus of the University of Massachusetts Lowell, having earned a B.S. in Computer Science in 2022, and has been working in the field of artificial intelligence and machine learning since the beginning of 2023.

Jean Seda is an AI/ML Engineer at Lockheed Martin. He holds a Bachelor of Science in Computer Engineering (BSCE) from the University of Central Florida. With deep expertise in large-language-model applications, full-stack software development, and DevOps automation, Jean has delivered advanced AI capabilities and mission-critical software solutions. He also supports a portfolio of AI research and development projects within the RMS Training, Logistics, and Simulation business.

Creating a Scalable Virtual Flight Instructor Using Large Language Models

Colin Sullivan, Kyle Tauzer
Lockheed Martin
Orlando, FL
colin.m.sullivan@lmco.com, kyle.a.tauzer@lmco.com

Mark Cavanagh, Christopher Lee, Jean Seda
Lockheed Martin
Orlando, FL
mark.cavanagh@lmco.com, christopher.1.lee@lmco.com, jean.seda@lmco.com

INTRODUCTION

As the demand for skilled pilots continues to accelerate, there is a growing need for innovative solutions that offer accelerated learning outcomes without requiring dramatic increases in instructional resources. Recent experiments have explored the integration of virtual instructors, or intelligent tutoring systems, with flight simulators to enable tailored feedback to students without the need for direct instructor oversight. While these initial efforts have shown promise, they are often limited by manually coded, rules-based architectures that lack the flexibility to scale across multiple aircraft, syllabus standards, simulator types, and levels of pilot expertise. This paper investigates the use of open-source large language models (LLMs), agentic artificial intelligence (AI), and retrieval augmented generation (RAG) to create a virtual instructor that can be used as an adaptable, flexible, and effective training solution.

EARLIER WORK

Early attempts to develop virtual instruction for simulators focused on rules-based analysis and the development of finite state machines (Stottler and Domeshek et al., 2005). In this approach, traditional training needs and cognitive task analyses were used to decompose maneuvers into rules that could be quantitatively evaluated with the simulator data. Simulation engineers or technicians collaborated with pilot experts to define simulation variables and calculate deviations from a standard to score for each maneuver, resulting in feedback to trainees. This process requires a spiral development process to iterate multiple times with pilot experts, as the pilot experts typically do not have sufficient depth in implementation of the simulation system, and the simulation engineers and technicians typically do not possess adequate pilot expertise (Stottler and Domeshek et al., 2005).

Many of these systems have been in use at a small scale for years, but expanding them to support additional aircraft and maneuvers is often prohibitively time-intensive. Rules-based state machines must be developed that handle variations for each aircraft, maneuver, environmental condition, fault conditions, Visual Flight Rules (VFR), Instrument Flight Rules (IFR), airspace/geographies, and more. The number of potential iterations quickly becomes insurmountable, posing a challenge to scaling to full training pipelines.

More recently, multiple authors have proposed systems that use machine learning (ML) to extract pilot knowledge from raw flight data, leveraging expert pilot runs to extract hidden features that can be used to evaluate trainee flights (Yang et al., 2021). Attempts to create these systems have been hindered by data availability and labeling challenges. In one example, T-6 aircraft data from the United States Air Force Pilot Training Next initiative was analyzed to determine if maneuvers could be extracted by ML algorithms. A significant challenge was that the data did not contain labels indicating pilot expertise or whether it was flown correctly. The study noted some success in cleaning the data using ML techniques but did not report progress in categorizing maneuvers or evaluating them effectively (Samuel et al., 2022). Other attempts involved manually collecting expert pilot data for comparison against student runs, but these suffered from similar scalability issues to the rule-based state machine approach, typically reporting a limited scope on basic maneuver evaluation (Guevarra et al., 2023).

SYSTEM OVERVIEW

Experiments were conducted that leverage LLMs and widely available documentation on pilot training to create an AI-powered instructor, improving system scalability and eliminating the need for large datasets or direct input from a

pilot subject matter expert (SME). The system ingests relevant documentation such as aircraft flight manuals, airmanship standards, and instructor notes and uses open-source LLMs to compare these documents against simulator variables extracted from a training run. This solution is rapidly scalable, leveraging existing manuals and documentation that are widely available and adaptable to various contexts.

The developed prototype is an open architecture software system meant to provide a highly scalable solution to virtual instruction for pilot training. The system: 1. Is agnostic to any specific simulation or simulator, so long as data telemetry about flight characteristics is outputted. 2. Receives data about specific aircraft, maneuvers, and airspace only from pre-existing documentation or summaries (i.e. does not require datasets of exemplar maneuvers or pre-labelled graded maneuvers). 3. Can make use of a wide range of LLMs within its architecture. As input, the system takes raw time series simulation variables, a prompt from the user for the maneuver represented in the data (for instance "Cessna 172 Landing"), and a set of instructional documentation from which to evaluate the maneuver, such as flight manuals and airmanship standards. Based on these inputs, the system parses the data and establishes the necessary prompts to LLMs to provide outputs such as free text after action reviews, suggested scoring breakdowns, and detailed analysis of simulation variables against the provided documentation. To enable the evaluation of additional aircraft or maneuvers, only supplementary documentation needs to be added to the RAG databases; no modifications to the underlying architecture are necessary. The resulting architecture is highly flexible, enabling the system to become knowledgeable in a field through uploading relevant documentation. This flexibility could also enable the system to be adopted to broader training use cases including medical debriefs or other knowledge-intensive domains outside of pilot training.

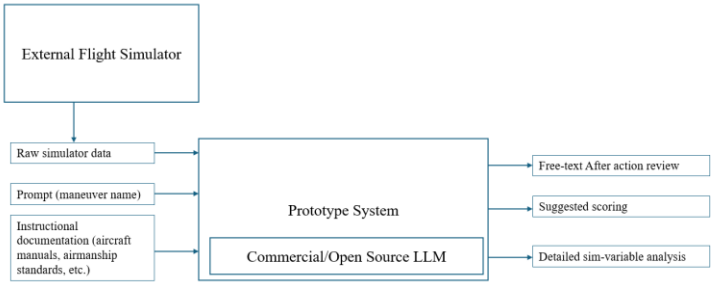

Figure 1: High Level Input / Output Diagram

Figure 2 shows screenshots from sections of the system output. By utilizing prompt engineering, experiments have been conducted with various output formats, including free form text and more granular analysis of each variable. The Future Work Section will describe a plan to format the system's output to be more in line with instructor feedback.

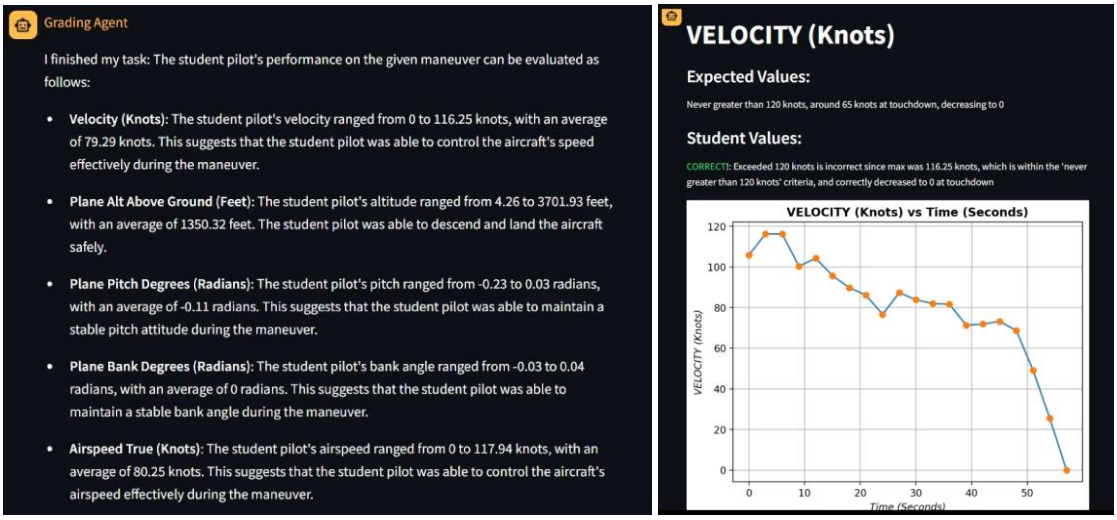

Figure 2: Example Output

Key Technical Challenges

Designing an LLM-based virtual instructor for flight data analysis proposes several technical challenges. First, most flight simulators output hundreds of variables of data at a frequency of approximately 30Hz, resulting in a dataset that far exceeds the context windows of most LLMs. The system will need to sample and filter the data, reducing its length while preserving sufficient information to effectively analyze the student's maneuver. Second, most flight simulators output tabular data, which is structured in a way that is not easily interpretable by LLMs. This tabular data must be processed and transformed into a format that can be easily understood by LLMs. Finally, the LLM must demonstrate a comprehensive understanding of how to perform complex flight maneuvers, which requires knowledge of the underlying dynamics and physics of flight. Open-source LLMs may not be equipped out of the box with such an understanding.

APPROACH

The developed architecture is composed of five main components: 1. retrieval augmented generation (RAG), 2. agentic AI, 3. opensource LLMs, 4. simulation data formatting, and 5. task decomposition. Each of these components are explained below.

Retrieval Augmented Generation (RAG)

RAG is a method used to expand the information available to LLMs by allowing for the retrieval of relevant documents or data from an external source. Instead of relying only on what the model was trained on, RAG fetches additional content at runtime and inserts this data into the LLM prompt, helping to improve the quality and accuracy of the responses. This framework was first introduced by Lewis et al. in 2020.

The system's implementation of RAG uses cosine similarity to identify relevant information from the document set. Cosine similarity is a method used to measure how similar two pieces of text are by comparing their vector representations. Stored documents, which include information about specific aircrafts and the maneuvers being assessed, are first converted into embeddings. Cosine similarity then identifies which sections of the documents are most relevant to the query. This helps ensure that the system provides accurate, context-aware responses. To support this retrieval process, the system relies on preprocessed documentation. For more detail on the documents and data RAG retrieves, please refer to the Experimentation Section.

Agentic AI

AI agents allow for the orchestration of different tasks within the solution. They enable multiple LLMs to work together toward the same goal. Each agent operates independently but can also communicate with other agents to share information or results. These agents can be equipped with tools such as document parsing, web search, or other custom functions making it possible to build more flexible and specialized solutions. To coordinate these agents effectively, the framework CrewAI is leveraged, which orchestrates the behavior and collaboration of multiple agents. The framework provides the tools needed to design complex agent workflows that are tailored to specific solutions. The system uses three main agents: 1. RAG Selection Agent, 2. Scenario Analysis Agent, and 3. Grading Agent.

The RAG Selection Agent determines which RAG documents should be used to answer the user's query. It compares the user's input to the available files, selecting the most relevant documentation for the task. Once the RAG content is retrieved, the Scenario Analysis Agent analyzes the returned data to determine which parts of the data, such as relevant columns or features, should be focused on. Its goal is to extract the most important information from the retrieved file based on the context of the query. Finally, the Grading Agent takes the analyzed content and evaluates it according to the specific rubric or criteria found in the RAG search. This could include scoring for accuracy, completeness, or how well it matches expected outcomes. Together, these agents work in sequence to mirror a structured decision-making pipeline: selecting data, analyzing it, and evaluating the result. This modular approach makes the system easier to manage, test, and improve over time. The tone and pacing of agents' responses are driven by prompt engineering and sampling flight data every two seconds. Agents are instructed to provide constructive and critical feedback on a student's performance, offering actionable insights and suggestions for improvement.

Open-Source Large Language Models (LLMs)

During the experiments, four open-source LLMs were tested. Table 1 lists key information on each LLM, along with their average score on the Massive Multitask Language Understanding (MMLU) benchmark. The MMLU tests LLMs on 57 subjects including mathematics, computer science, physics, and more.

Model	Company	Parameters	Context Window (Tokens)	MMLU Score (All Subjects)
Mixtral 8x22B Instruct v0.1	Mistral AI	8x22B	64k	77.80%
LLaMA 3.1 (70B)	Meta	70B	128k	80.10%
LLaMA 3.1 Nemotron	Nvidia Fine-Tuned	70B	128k	80.20%
LLaMA 3.3 (70B)	Meta	70B	128k	79.10%

Table 1: Tested LLMs on MMLU

Open-source LLMs were utilized for testing as these models can be hosted internally, allowing for them to ingest propriety and export-controlled information. Additionally, they provide the flexibility to fine-tune or further customize parameters in the future. The system is designed to be LLM-agnostic, allowing for integration with most LLMs. As newer ones become available, they can be incorporated into the system with minimal additional development.

Formatting of Simulation Data

To convert the tabular simulation output to a format that can be understood by LLMs, multiple formatting methods were experimented with. These methods included parsing the data into python dictionaries, JSON arrays, and generating a stat analyzer. The python dictionaries were generated by mapping aircraft variables to a list of integers. The JSON arrays were generated by listing variable values for each timestamp. Finally, the stat analyzer was designed to help the LLMs with basic math by calculating statistics like maximums, minimums, and range. To address the LLMs' limited context window, the parser samples data every 2 seconds and only parses columns returned by the Scenario Analysis Agent, which was described in the RAG Section. Additionally, the parser also converts all values to the correct units.

Format	Example
Dictionary	{"TIME (S)": [0, 2, 4, 6, 8], "VELOCITY (Kts)": [94, 100, 111, 111, 108], "VERTICAL SPEED (Ft/s)": [0, -4, 4, 14, 18], "ALT (Ft)": [3839, 3779, 3607, 3023, 3010], "PITCH (°)": [-11, -9, -7, -4, -3]}
JSON	{"TIME (S)": 0, "VELOCITY (Kts)": 94, "VERTICAL SPEED (Ft/s)": 0, "ALT (Ft)": 3839, "PITCH (°)": -11}, {"TIME (S)": 2, "VELOCITY (Kts)": 100, "VERTICAL SPEED (Ft/s)": -4, "ALT (Ft)": 3779, "PITCH (°)": -9}, {"TIME (S)": 4, "VELOCITY (Kts)": 111, "VERTICAL SPEED (Ft/s)": 4, "ALT (Ft)": 3607, "PITCH (°)": -7}, {"TIME (S)": 6, "VELOCITY (Kts)": 111, "VERTICAL SPEED (Ft/s)": 14, "ALT (Ft)": 3023, "PITCH (°)": -4}, {"TIME (S)": 8, "VELOCITY (Kts)": 108, "VERTICAL SPEED (Ft/s)": 18, "ALT (Ft)": 3010, "PITCH (°)": -3}
Stat Analyzer	{ "VELOCITY (Kts)": {"values": [94, 100, 111, 111, 108], "min": 94, "max": 108, "average": 104.8, "range": 14, "delta": 14}, "VERTICAL SPEED (Ft/s)": {"values": [0, -4, 4, 14, 18], "min": -4, "max": 18, "average": 8, "range": 22, "delta": 18}, "ALT (Ft)": {"values": [3839, 3779, 3607, 3023, 3010], "min": 3010, "max": 3839, "average": 3451.6, "range": 829, "delta": 829}, "PITCH (°)":{"values": [-11, -9, -7, -4, -3], "min": -11, "max": -3, "average": -6.8, "range": 8, "delta": 8}}

Table 2: Data Formats

Task Decomposition

For the task decomposition experiment, the Grading Agent's single task was broken into multiple smaller, more manageable sub-tasks. More specifically, instead of presenting the Grading Agent with the comprehensive dictionary containing flight data and variables, each variable's values were sent to the Grading Agent one at a time. For example, if there were 15 values that needed to be analyzed, the Grading Agent would be called 15 times, once for each variable.

This decomposition strategy aims to reduce the cognitive load on the Grading Agent, allowing it to concentrate on a single variable at a time.

Architectural Design

Figure 3 illustrates the system architecture, which is designed to analyze one maneuver at a time. First, students fly a maneuver in their flight simulator of choice and save the raw flight data output, which is typically stored as CSV files. As input, the system takes this raw flight data, along with a text string indicting the aircraft and maneuver performed (i.e. "Cessna Landing"). The RAG Selection Agent then retrieves the relevant data from the vector databases to serve as the foundation of the LLM's response. The Scenario Analysis Agent uses the returned RAG data to identify the columns in the flight data to parse. The Parser extracts the relevant columns, transforms the data into a suitable format for the LLM, and converts variables to the correct units. Finally, the Grading Agent analyzes the parsed data in conjunction with the RAG output to assess the student pilot's performance.

Figure 3 does not depict the task decomposition experiment, but the architecture remains largely unchanged with the only modification being that the Grading Agent would be invoked multiple times to analyze each variable individually.

Student First Flies Scenario in Flight Simulator

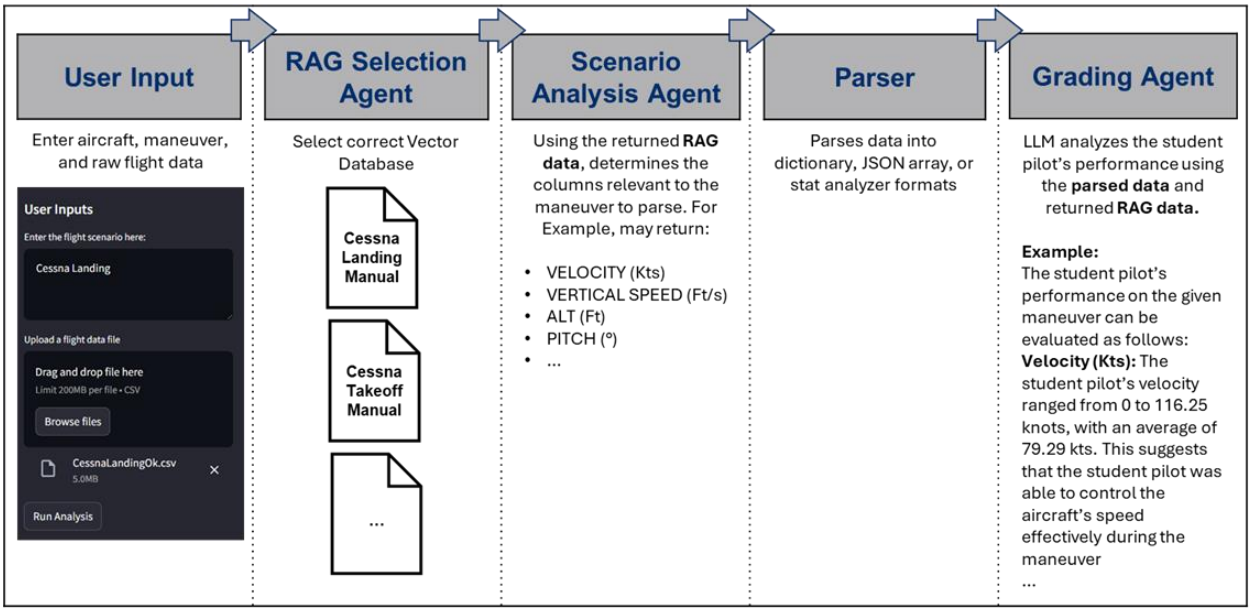

Figure 3: System Architecture

EXPERIMENTS

Data

The RAG vector databases are populated with flight manuals and pilot information from a range of authoritative sources, primarily from the Federal Aviation Administration (FAA) and Cessna. The experiments only analyzed Cessna maneuvers, but other aircrafts can be supported by adding their respective manuals. These documents are preprocessed to extract relevant information, which is then incorporated into the RAG vector database. This knowledge base provides a comprehensive foundation for the system to draw upon when evaluating pilot performance and providing feedback. The documents used to populate are listed in Table 3 below.

Document	Source
Cessna Model 172S Information Manual	Cessna
Aeronautical Information Manual: Official Guide to Basic Flight Information and ATC Procedures (AIM)	FAA
Airplane Flying Handbook	FAA
Aviation Instructor's Handbook	FAA
Pilot's Handbook of Aeronautical Knowledge	FAA
Instrument Flying Handbook	FAA
Private Pilot – Airplane Airman Certification Standards	FAA

Table 3: Document Data Uploaded to RAG

While the system has been designed to work with any flight simulator, the experiments used Prepar3D® (P3D). P3D provides a realistic simulation environment with a comprehensive set of over 600 simulation variables. These variables encompass various aspects of flight, including aircraft dynamics, navigation, weather, and system performance. During flights, these simulation variables are outputted at 30Hz to a CSV file, where each column represents different variables and each row includes its value at a given timestamp. The initial prototype only analyzes aircraft data and does not account for environmental factors such as wind and temperature. The variables extracted from P3D include:

Variable Types	Examples
Kinematic and Dynamic Parameters	Velocity, Acceleration, Rotation
Aircraft State and Configuration	Altitude, Airspeed, Control Surface Positions
Engine and System Performance	Throttle, RPM, Fuel
Warning and Indicator Systems	Stall, Overspeed
Pilot Inputs and Control Surface Positions	Yoke, Rudder Pedal

Table 4: P3D Flight Data Variables

Evaluation Method

To evaluate system performance, takeoff and landing maneuvers were flown in a Cessna 172 using P3D. These maneuvers were selected for evaluation because of their complexity, which require a sophisticated understanding of aircraft dynamics and flight procedures to accurately assess. Table 5 outlines additional information on each scenario flown.

Maneuver	Aircraft	Flight Length (Seconds)	Description of Student's Performance
Takeoff	Cessna 172	91	Good takeoff, only issues are: • **Vertical Speed:** Exceeded • **Flaps:** Not Deployed • **Max Engine RPM:** Exceeded
Landing	Cessna 172	201	Average landing, issues include: • **Pitch:** Exceeded • **Vertical Speed:** Exceeded • **Brakes:** Not Deployed • **Max G Force:** Exceeded • **Spoilers:** Not Deployed

Table 5: Description of Scenarios Flown

Each model was evaluated over 70 iterations for both landing and takeoff missions, resulting in a total of 140 runs per model. The iteration count was determined by observing convergence of average scores for each maneuver at 70 runs.

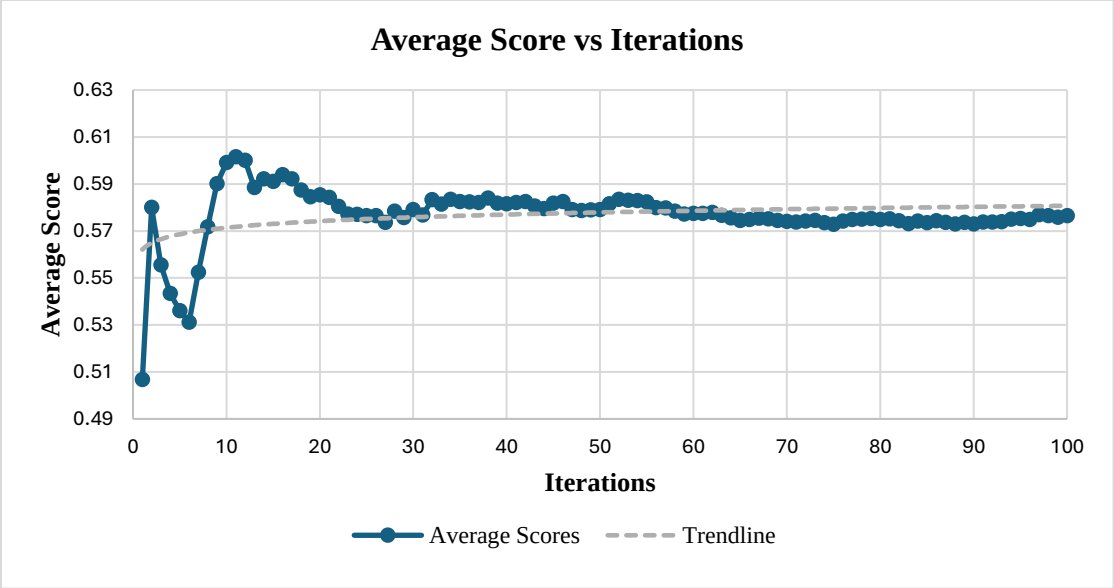

Table 6: Average Scores During Cessna Takeoff Mission, Showing Score Convergence Around 70 Iterations

The Grading Agent formatted its answer as follows: {Selected Variable: [Expected lower bound value, expected upper bound value, whether the student was within these bounds]}. A python script utilized a predefined rubric to grade each run, ensuring consistency and objectivity in the evaluation processes. The rubric is shown in Table 7 below.

Criteria	Points
Selects the Correct Vectorized RAG Document	15
Selects Relevant Variable Column for Given Maneuver	1
Lower Bound is Within 10% of the Actual Lower Bound	1
Upper Bound is Within 10% of the Actual Upper Bound	1
Correctly Determines if the Student is Within the Correct Bounds	1
Total Possible Points	**15 + 4 × len(relevant columns)**

Table 7: Evaluation Rubric

Results

The baseline model consists of the Mixtral 8x22B v0.1 LLM, RAG, and a data formatting pipeline that organizes parsed data into a dictionary structure. Models 0 through 3 have the same architecture but utilize different LLMs for the AI Agents. After determining the best LLM for the tasks, different data formatting and architecture experiments were run in models 4 through 6. Table 8 below lists the results of the experiments.

Model	LLM	Architecture	Average Score (%)	Average Time (S)
0 (Baseline)	Mixtral 8x22B v0.1	• RAG • Dictionary	70.52	20.15
1	LLaMA 3.1 (70B)	• RAG • Dictionary	64.80	30.46
2	LLaMA 3.1 Nemotron	• RAG • Dictionary	69.97	**17.83**
3	LLaMA 3.3 (70B)	• RAG • Dictionary	**71.16**	38.00
4	LLaMA 3.3 (70B)	• RAG • Dictionary • Stat Analyzer	70.99	31.92
5	LLaMA 3.3 (70B)	• RAG • JSON	70.23	41.99
6	LLaMA 3.3 (70B)	• RAG • Task Decomposition	67.17	55.41

Table 8: Results

ANALYSIS

LLMs Affecting System Performance

The primary factor influencing system performance was the LLM used by the AI agents. LLaMA 3.1 (Model 1) exhibited the worst performance, followed by LLaMA 3.1 Nemotron, Mixtral 8x22B v0.1, and ultimately LLaMA 3.3 (Model 3) demonstrated the best performance. This ordering is mostly consistent with the chronological development of the models, with LLaMA 3.3 being the newest. This correlation suggests that the system's design, which allows for integration with any LLM, will likely continue to benefit from advancements in LLM technology and lead to improved performance as newer and more sophisticated models become available. LLaMA 4 was recently released by Meta and will soon be tested with the system to determine if performance continues to improve.

Data Formatting

The formatting of the parsed data sent to the AI agents had a significant effect on system performance. The dictionary resulted in the best performance, with the stat analyzer following close behind. It was hypothesized that the stat analyzer's ability to automate the calculation of key values would improve system performance, however the results showed little change in final average score.

The selection of dictionary and JSON formats as data representation schemes was motivated by their ubiquity in programming and the fact that LLMs are often trained on extensive coding datasets. The dictionary format (Model 3) outperformed the JSON format (Model 5) by nearly 1%. While this difference is small, the disparity may be contributed to the concise nature of the dictionary. As shown earlier in Table 2, the dictionary format contained only 198 characters while the equivalent data in JSON was more than twice as long at 512 characters. This increased verbosity may have introduced additional complexity and noise, ultimately hindering the model's ability to extract relevant information.

Task Decomposition

The experiments revealed that task decomposition hindered system performance, with model 6 performing about 4% worse than model 3. It was hypothesized that decomposing the task into smaller, more manageable sub-tasks would enhance performance as each agent would manage a more straightforward task. However, the results suggest that the task decomposition may have been overly granular, resulting in the system losing the broader context of the flight. This loss of context may have impeded the system's ability to make accurate determinations, as the grading agent was only considering individual variables in isolation, rather than as part of a larger, interconnected system. This highlights the importance of having a balance between task decomposition and contextual understanding.

Time to Complete Tasks

Running LLaMA 3.3 with RAG and the dictionary (Model 3) format averaged around 31.92 seconds to complete tasks. In contrast, the JSON format (Model 5) averaged around 41.99 seconds, or more than 10 seconds longer. As mentioned earlier, this difference is likely due to the concise nature of the dictionary compared to the longer JSON format, which likely introduced additional computational time to analyze the scenario. Finally, the task decomposition architecture (Model 6) resulted in the longest runtime at around 55.41 seconds. This is expected, as this architecture involves more LLM calls as each variable is analyzed independently.

Common Errors

The system exhibited three common errors in its output: hallucinations, a difficulty distinguishing between negative and positive values, and incorrect math calculations. First, the system was prone to hallucinations, where incorrect expected values were generated, leading to misleading feedback. In one example, the RAG documentation indicated that the expected values for velocity should be between 0 and 120 knots, but the system outputted 0 and 65 knots; 65 was not included anywhere in the RAG data. Second, the system struggled to distinguish between negative and positive numbers. For example, if the RAG document indicated that the aircraft should decrease its vertical speed by around 7.5 feet per second, the system often failed to recognize that this meant that the student's value should be around -7.5, not +7.5. Finally, the system was also susceptible to common math errors, such as failing to recognize whether a student's value was within or outside the acceptable range. Surprisingly the stat analyzer did not have a major effect on preventing these common math errors.

FUTURE WORK

Improve System Accuracy on Common Errors

Future work will focus on addressing the errors listed earlier. The hallucinations error has the largest effect on system performance, as it can be challenging for student pilots to recognize when the output has incorrect information. To mitigate hallucinations, experiments will be conducted into improved prompt engineering and RAG parsing techniques. Specifically, the current cosine similarity-based RAG implementation will be compared to alternative distance metrics, including Euclidean Distance, Manhattan Distance, and Dot Product. To address the system's difficulty in distinguishing between negative and positive values, the vector databases will be updated to include explicit sign indicators for values. To address math errors, future experiments will explore leveraging improved parsing techniques and agents using external math tools to validate their calculations.

Tailor Responses to Instructor Feedback

Work has begun collaborating with pilot SMEs to refine the system's output so that it aligns more closely with the style and tone of an actual instructor. Data has been collected from a SME pilot where they have assessed the performance of multiple recorded flights. Prompt engineering techniques are currently being leveraged to enhance the outputs of the system, aiming to make it similar in style and content to a pilot instructor's evaluations.

To determine the similarity between the system's output and that of a human instructor, Likert-scale learning ratings and manual human instructor reviews will be utilized. The system will also continue using the main evaluation method described earlier in the paper. While the evaluation method described earlier allows for an automated and objective

assessment of system accuracy, it does require generating a grading rubric for each flight which limits its scalability. Likert-scale learning ratings and human reviews also suffer from similar scalability issues. To solve this problem, investigations are underway into utilizing LLM as a judge to analyze future outputs faster.

While tailoring the output to instructor feedback, additional safeguards will be added to prevent inappropriate or off-topic dialogue generation. Prompt engineering is currently used to keep the system on topic. Future methods including input and output filters that analyze inappropriate content, retokenizing inputs to reduce prompt injection attacks, and additional human verification of system input and output are also being investigated.

Fine-Tuning

Prompt engineering alone may be insufficient to fully capture the nuances and formatting of instructor feedback. To further improve the system's performance, investigations are underway into fine-tuning the model on a small dataset of SME evaluations using few-shot learning techniques. This involves training the model on a limited number of examples, enabling it to learn patterns and relationships in the data. Few-shot learning has proven effective in adapting language models to new tasks and domains with limited data.

The goal is to adapt the system to produce outputs that are similar in style and content to those of the SME evaluators. Achieving this goal will enable the model to provide more accurate and relevant feedback to pilots, ultimately improving the effectiveness of the system.

Maneuver Detection Work

Currently the system can only evaluate one flight maneuver at a time. In a traditional training exercise, pilots fly a sortie of multiple maneuvers that are then all evaluated by an instructor. Work has already begun to allow the system to automatically detect maneuvers flown in a sortie and then analyze the performance of each. This would also allow the system to analyze much longer scenarios. The current approach to maneuver detection has shown promise, but its performance has not been thoroughly evaluated against ground truth annotations from SMEs. Although the system can detect certain maneuvers with reasonable accuracy, its consistency and reliability require improvement. To address these limitations, several potential avenues for enhancement are being explored, including refining the maneuver definition information in the RAG database, a critical knowledge source for the system. Additionally, alternative approaches to maneuver detection may be investigated, such as incorporating additional data sources or features, or leveraging machine learning-based methods.

Another potential technique to improve maneuver detection work would be to develop a data strategy that enables the collection and annotation of a large dataset of flights with SME evaluations. This dataset will serve as a benchmark for evaluating the performance of the system and other maneuver detection systems, and will provide a foundation for exploring new and innovative approaches to maneuver detection. With a sufficiently large and diverse dataset, there are opportunities for transfer learning, domain adaptation, and the development of more advanced and sophisticated maneuver detection systems. Future research in this direction has the potential to significantly improve the accuracy, consistency, and reliability of maneuver detection in the system, ultimately contributing to the broader goal of developing more effective and efficient pilot training systems. A key challenge of this approach will be maintaining the system's flexibility and scalability to accommodate new aircraft and maneuvers with minimal additional effort.

CONCLUSION

The best model consists of an agentic workflow powered by LLaMA 3.3 (70B), RAG, and simulation data parsed into a dictionary structure. This model achieved a 71.16% accuracy and took an average of 38.00 seconds to run. While further work is required to improve system accuracy and format the output in a manner more consistent with instructor feedback, it has been demonstrated that LLMs can be used to analyze raw flight simulator data to provide robust analysis of student performance on complex missions. Additionally, as newer LLMs become available, the system's accuracy will continue to improve without requiring additional modifications to the underlying architecture.

REFERENCES

Guevarra, M., Das, S., Wayllace, C., Demmans Epp, C., Taylor, M. E., & Tay, A. (2023). "Augmenting Flight Training with AI to Efficiently Train Pilots." In Proceedings of the *Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)*.

Lewis, Patrick, et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *Advances in neural information processing systems* 33 (2020): 9459-9474.

"meta-llama/Llama-3.1-70B-Instruct · Hugging Face." "*meta-llama/Llama-3.1-70B-Instruct · Hugging Face*, https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct. Accessed 2 June 2025.

"meta-llama/Llama-3.3-70B-Instruct · Hugging Face." "*meta-llama/Llama-3.3-70B-Instruct · Hugging Face*, https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct. Accessed 2 June 2025.

"Mistralai/Mixtral-8X22B-Instruct-v0.1 · Hugging Face." *Mistralai/Mixtral-8x22B-Instruct-v0.1 · Hugging Face*, huggingface.co/mistralai/Mixtral-8x22B-Instruct-v0.1. Accessed 2 June 2025.

"nvidia/Llama-3.1-Nemotron-70B-Instruct-HF1 · Hugging Face." *nvidia/Llama-3.1-Nemotron-70B-Instruct-HF1 · Hugging Face*, https://huggingface.co/nvidia/Llama-3.1-Nemotron-70B-Instruct-HF. Accessed 2 June 2025.

Samuel, K., LaRosa, M., McAlpin, K., Schaefer, M., Swenson, B., Wasilefsky, D., Wu, Y., Zhao, D., & Kepner, J. (2022). "AI Enabled Maneuver Identification via the Maneuver ID Challenge." In Proceedings of the *2022 Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*.

Stottler, D., & Domeshek, E. (2005). "Intelligent Tutoring Systems (ITSs): Advanced Learning Technology for Enhancing Warfighter Performance." In Proceedings of the *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2005* (Paper No. 2112).

Yang, S., Yu, K., Lammers, T., & Chen, F. (2021). "Artificial Intelligence in Pilot Training and Education - Towards a Machine Learning Aided Instructor Assistant for Flight Simulators." In *Communications in Computer and Information Science* (pp. 1-12). doi: 10.1007/978-3-030-78642-7_78