Towards Naturalistic Human-Machine Teaming with LLM Agents: A Case Study in Air Traffic Management

Nathan Xue[*,**]
Purdue University, West Lafayette, Indiana 47907, USA

Dhriti Verma[*,†]
Cornell University, Ithaca, NY 14850, USA

Vriksha Srihari[*,‡]
Georgia Institute of Technology, Atlanta, GA 30332, USA

Wiktor Piotrowski[§] and Kenny Chour[¶]
Metis Technology Solutions, NASA Ames Research Center, Moffett Field, CA 94035, USA

Krishna Kalyanam[‖]
NASA Ames Research Center, Moffett Field, CA 94035, USA

Natural language is the dominant form of interaction modality in Air Traffic Management (ATM), currently operated by human controllers. However, most Artificial Intelligence (AI) research in this domain uses low-level interfaces that are disconnected from how controllers communicate. This paper introduces an LLM-manned team framework: multiple large language model (LLM) agents assuming specialized ATM roles that collaborate via natural language within the open-source BlueSky simulator environment. We instantiate two roles: (1) a communication agent that translates controller-style natural language commands using retrieval-augmented generation, and (2) a deconfliction surrogate fine-tuned with reinforcement learning to propose collision-avoidance maneuvers in horizontal conflict scenarios. The interface role achieves 71% translation accuracy, while the deconfliction agent achieves a success rate up to 96.1% from 65.9% compared to baselines. These results demonstrate the promising potential for LLM-manned teams to serve as decision-support tools and testbeds for human-machine teaming in ATM.

I. Introduction

Human-Machine Interaction (HMI) is a crucial field of study in a wide range of applications, including aviation, healthcare, and manufacturing. The goal of HMI research is to develop systems that are intuitive, efficient, and safe for users. In the Air Traffic Management (ATM) domain, there is a growing need for intelligent HMI systems capable of understanding and responding to natural language inputs in real-time. This demand is partly driven by the projected increase in air traffic volume [1] and the emergence of new types of aircraft, such as eVTOLs (electric vertical take-off and landing) and UAS (unmanned aerial systems) under various Advanced Air Mobility efforts. Novel HMI systems may significantly alleviate human workload by directly interpreting natural language (NL) inputs and automatically generating understandable responses, or executing complex functions without having to resort to low-level programmatic interfaces. As it stands, the existing Air Traffic Management system is not well equipped to support control via natural language commands, nor are there many instances of intelligent decision-making autonomous systems compatible with such natural language requirements.

[*]Equal Contribution
[*]NASA OSTEM Intern, Aerospace Engineering, Purdue University.
[†]NASA OSTEM Intern, Operations Research and Engineering, Cornell University.
[‡]NASA OSTEM Intern, Robotics, Georgia Institute of Technology.
[§]Principal Research Engineer, Metis Technology Solutions, NASA Ames Research Center.
[¶]Senior Software Engineer, Metis Technology Solutions, NASA Ames Research Center, AIAA Member.
[‖]Deputy Director, NASA Aeronautics Research Institute, NASA Ames Research Center, AIAA Associate Fellow.

Recently, Large Language Models (LLMs) have demonstrated natural language understanding capabilities that surpass traditional rule-based approaches, making them suitable for tasks such as natural language processing, question answering, and text generation. Exploring LLMs to enhance human-machine interactions has become a very popular field of research in recent years. Beyond natural language processing tasks, LLMs have also been used to develop embodied AI systems that are capable of decision-making [2] or translating natural language instructions into executable code, making them a valuable tool in domains such as robotics [3] and games [4]. However, there are still concerns about hallucinations in LLM responses, in part due to their inability to access up-to-date information. To mitigate these drawbacks, techniques such as *Retrieval Augmentation Generation* [5] (RAG) and *tool-learning* [6] have been proposed to augment LLM agents with external knowledge sources for practical task-solving. Furthermore, LLMs can be fine-tuned with domain-specific knowledge using supervised or reinforcement learning. In the ATM field, LLMs have been used for decision-making [7–9], information retrieval [10], and classification [11, 12]. Despite significant uptake in the use of LLMs in ATM, there remains a critical need to develop intuitive human-machine interfaces using LLMs, particularly for many of the complex problems ATCs (Air Traffic Controllers) face, such as deconfliction.

To address this need, we first propose a framework that provides a natural language interface that seamlessly integrates with existing ATM simulation systems, such as BlueSky [13], a highly configurable open-source air traffic simulation environment. In this way, the framework aims to reduce the learning curve for novice users. Specifically, our proposed approach leverages the tool-calling capabilities of LLMs to convert natural language queries into executable commands in the application. Our framework leverages the simulator's API structure by processing documentation.

Furthermore, human-in-the-loop systems are crucial for safety-critical systems in domains such as air traffic management. However, human involvement typically renders the simulations as very time-consuming and expensive (especially when working with aviation subject matter experts). To that end, we also introduce LLM-based agent surrogates that aim to emulate the decision-making processes of human air traffic controllers and are capable of expressing these decisions in natural language. These surrogates are intended for use in simulation, training, and prototyping of human-machine teaming concepts, not as a replacement for certified controllers in operational settings. In contrast to existing ATC AI agent research, we demonstrate the capabilities of small language models, such as GPT-2 [14], as evidence that LLMs can be useful proxies for interpretable decision-making. Through reinforcement learning-based LLM fine-tuning, our approach is efficient and fast to train, while exhibiting significantly improved performance in multi-agent aircraft deconfliction scenarios.

Overall, our contributions are twofold:

1) We demonstrate the effectiveness of combining RAG and tool-calling capabilities with open-source LLMs to create a user-centric, natural language interface to the BlueSky simulation environment, enabling an LLM communication role within an LLM-manned team.
2) We showcase the composability of this approach with an upstream LLM-based deconfliction surrogate, highlighting how multiple LLM agents with ATC-relevant roles can collaborate within a single framework to support complex ATM scenarios.

II. Related Works

There is extensive literature discussing the many roles of generative AI in aviation [15], ranging from general-purpose AI assistants, through document and information retrieval, to complex reasoning and real-time decision-making.

In [7], the authors propose an approach to using large language models (LLMs) to assist traffic managers in summarizing Ground Delay Program information. They investigate both in-context learning as well as fine-tuning for summarization tasks. In [16], a general-purpose LLM, AviationGPT, was created by pre-training and fine-tuning by considering aviation documents. AviationGPT assists with document summarization, information extraction, question-answering, report-querying, as well as data exploration and cleaning. To improve the accuracy and correctness of LLM-based systems, a RAG approach can be used to process and utilize datasets and documents. Yang et al. [17] leverage a RAG-based approach to analyze aviation accident reports to gain important insights into airspace safety. AeroQuery RAG and LLM [10] is another example of a RAG-based aviation LLM that can answer queries on a broader range of aerospace topics, such as industry standards, protocols, certifications, and designs.

Interacting with external software systems, such as simulation environments, requires conforming to their syntax rules and rigid structures. Currently, LLMs do not have an innate ability to dynamically interact with external applications unless done via tool-calling [18] (also called function-calling). Instead of generating text, tool-calling enables LLMs to interact with external systems by executing predefined snippets of code (functions/tools) that perform specialized tasks or dynamically retrieve data. Tools are typically defined manually, but there are many tool-learning approaches that

automatically compose such functions from data and/or interactions [19–23]. To date, tool-calling and tool-learning are significantly under-researched and underutilized in the aviation domain [24].

Efficient decision-making is a crucial aspect of the aviation domain, and there has been significant research focus on leveraging LLMs for that purpose. In [8] a reinforcement learning framework was developed to assist pilots in making decisions during flight operations. In [25], a recommender system is built using GPT-4o to help select a flight operator's preferred flight plan based on natural language.

Interestingly, LLMs have also been used for deconfliction tasks as opposed to more traditional RL approaches [9, 26, 27]. In [24], the authors develop a tool-calling framework that integrates with the BlueSky simulator to deconflict aircraft. Up to 4 aircraft were tested for a variety of tactical scenarios, including crossing, head-on, parallel, and converging conflicts. The authors found that the best performing model was GPT-4o configured in a single-agent manner versus Llama-70B. However, there is little say in regards to the solution quality of vehicles in terms of distance traveled or power consumption. In a similar vein, LLMs have also been considered in the robotics domain, particularly in the Multi-Agent Path Finding (MAPF) community [28], which is highly relevant to trajectory planning in the aviation domain. However, these works did not explore the potential to improve LLM decision-making through fine-tuning, which could further justify LLMs' potential in modernizing ATC.

III. Approach

A. Overview

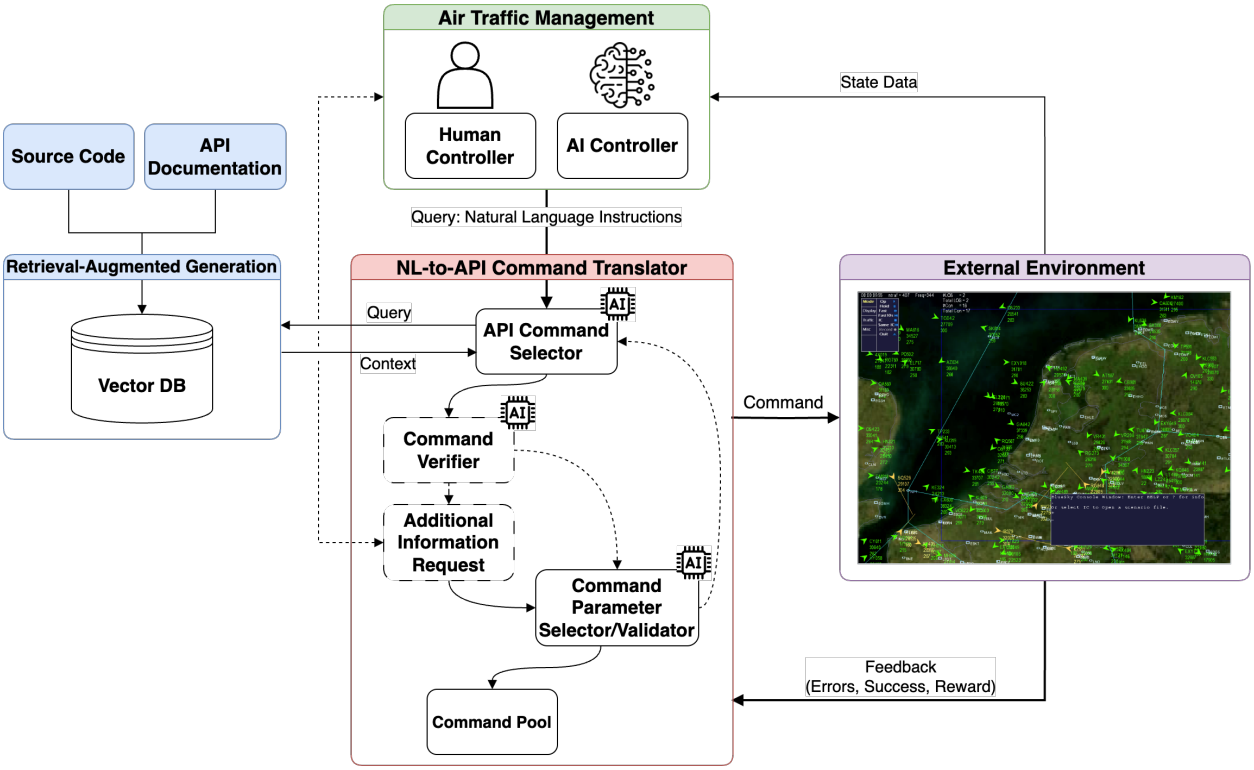


Fig. 1 High-level architecture of the proposed framework for the NLP translator. LLM modules/agents are denoted by an *AI* logo in their upper right corner. The dashed boxes are optional components in this study.

The proposed framework treats air traffic management as a collection of modular roles that can be instantiated as LLM-based agents operating over a shared interface to a simulation environment. Each agent assumes a specific responsibility traditionally handled by human controllers, such as issuing tactical clearances, managing traffic flows, or resolving conflicts, and interacts with other agents and the simulator through the natural language translator. In this study, we consider a single ATC surrogate agent that generates control decisions based on traffic state, specifically

horizontal aircraft deconfliction. Natural language commands are routed from the ATM-specific agents through the NL-to-API translator agent, which translates controller-style utterances into executable BlueSky commands.

The architecture is deliberately modular so that individual agents can be replaced or upgraded without redesigning the entire system. As long as the input-output contracts are preserved, different foundation models, retrieval strategies, or control policies can be plugged in, enabling systematic experimentation and incremental expansion to additional ATC functions.

B. Natural Language Translation

In a pre-processing step, the RAG module generates a vectorized database from the provided API documentation of the external system (i.e., BlueSky). In essence, the database contains a collection of chunks of text that describe every function in the API. These descriptions will be analyzed to best select the appropriate function from the API, which will be used as an extended context for each user query. LLMs continue to improve their understanding of data beyond natural language, as seen by the emergence of numerous generative AI coding assistants, such as Microsoft's Copilot. In the absence of adequate API documentation for the external system, our framework could potentially also leverage the external system's source code (if available).

The process begins with a query expressed in natural language by a human or AI user. The query contains instructions on what the user wants to happen in the execution environment. In BlueSky, these could include inserting a new aircraft, adjusting existing aircraft's trajectory or flight parameters, speeding up the simulation, etc. Once the query has been submitted, it is then fed into the RAG module, which selects the relevant additional context based on the top K most relevant contents from the database generated in the pre-processing step. Using RAG mitigates the widely known LLM hallucination issues by forcing it to directly reference the provided material (i.e., relevant parts of the API documentation) and prioritize it over its internal pre-trained data. Given the context, the first LLM agent selects the appropriate API function and sends the output through a verifier agent that checks for any syntax errors (e.g., malformed function name, function that is not in the provided API). The user may be prompted to provide additional information if the initial command generation attempt is erroneous (e.g., missing aircraft callsign/ID). Once a valid command is selected, it is then passed to the next LLM agent, which validates conformity of the function arguments/parameters and fills in the blanks, if any. To ensure valid function invocation, the process will generate multiple outputs in an iterative fashion up to a preset number of times in the LangGraph memory object. The specific prompts for each agent are detailed in the Appendix.

The LLM outputs the command to be executed by the external system in a format specified by the API documentation. Note that, unlike existing works, the tools are not "hard-coded". Rather, the LLM directly outputs the command as a text string that can then be directly executed by the external system (BlueSky). To improve the performance of the LLM, it is provided with feedback from the external system (e.g., error trace, updated state variables post-function execution, reward signal) to refine its generation and improve its accuracy. This process is iterative to ensure successful execution (in the event of an error) and accuracy of the executed parameterized commands.

C. Aircraft Deconfliction

As part of the ATC surrogate agent, we show how a small pretrained language model (GPT-2) can effectively emulate controller-like deconfliction decisions in a dynamic air traffic environment using feedback from the environment and natural language prompts. The core idea is to fine-tune the language model using reinforcement learning, allowing it to issue safe and goal-directed control decisions in a structured aviation domain like BlueSky Gym.

1. Environment

We use the Horizontal Collision Resolution (HorizontalCREnv) scenario from the BlueSky-Gym simulator [29], shown in Figure 2. In this 2D environment, the ownship aircraft must reach a target waypoint while avoiding collisions with intruders. To achieve its task, the controller (human or AI) must dynamically adjust the heading of the ownship aircraft at each time step by up to $\pm 45^\circ$ from its current heading. Intruders have randomly generated speeds, headings, and positions, which are largely influenced by a randomly generated time until loss of separation (TLOS). The controller's decision-making is based on an environment-provided set of observations, including relative distances, bearing angles, and velocity components. The number of intruders is 5 per episode. An episode in the environment terminates only when the ownship reaches the target waypoint.

In this work, the agent is only invoked for conflict resolution, assuming conflicts are already known. This leaves

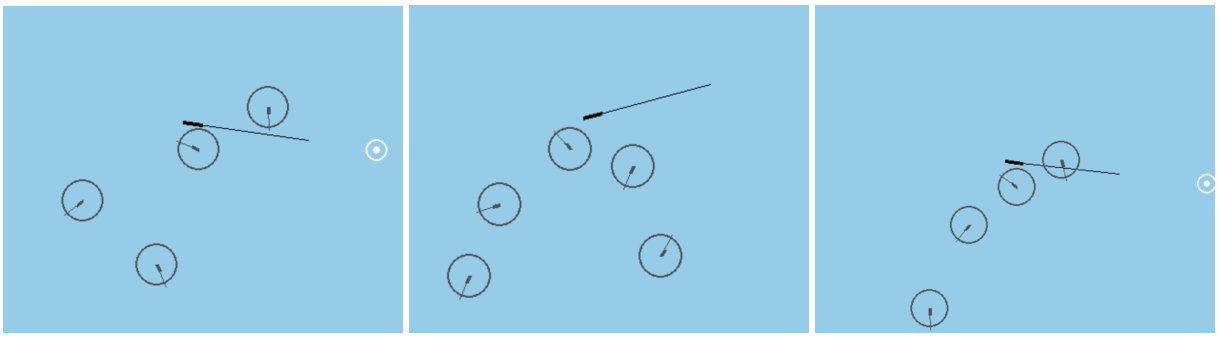


Fig. 2 Sample scenarios from the Horizontal Conflict Resolution BlueSkyGym environment. Ownship (central needle) must navigate to the target waypoint (white concentric circles) while avoiding intruder aircraft (black circles).

conflict detection as a separate subtask. The conflict detection process is performed by projecting the ownship's velocity vector forward within a specified lookahead time toward the waypoint and manually checking for conflicts with intruders in the vicinity. If no conflict is detected, the ownship automatically proceeds to the waypoint without calling the agent.

2. Reward Function

The reward function at each time step t is defined as follows:

$$r_t = r_{\text{intrusion}} + r_{\text{drift}} + r_{\text{hdg}} \tag{1}$$

1) **Intrusion term**: to heavily penalize an intrusion or provide a small reward for avoiding an intrusion. The intrusion penalty/reward is defined in Eq. 2:

$$r_{\text{intrusion}} = \begin{cases} -1 + w_i * |\frac{\text{CPA}}{\text{min dist}}|, & \text{if intrusion detected} \\ 0.02, & \text{otherwise} \end{cases} \tag{2}$$

where a large penalty is applied if an intrusion occurs, and a small reward is given when there is no intrusion. If there is an intrusion, there is an additional positive term added to the large penalty. The motivation behind this term is to provide a slight incentive for the agent to modify its heading to prevent the agent from maintaining its current heading when it detects an intrusion that appears "unavoidable" within a single time step. Without this term, the agent often chooses not to maneuver. Because it deems the scenario as unresolvable, it decides to minimize its losses by maintaining its heading and accepting the intrusion penalty while lessening the heading change penalty. By adding this small positive contribution, the reward encourages the agent to attempt incremental heading changes, which may ultimately resolve the intrusion across several time steps. w_i is the weighting given to this term (currently 0.4), and the term is related to the closest point of approach (CPA) divided by the minimum allowable loss of separation. Thus, larger heading changes in this particular situation (yielding a larger CPA) will minimize the intrusion penalty.

2) **Drift term**: to penalize deviation from the waypoint's bearing. Eq. 3 describes this penalty:

$$r_{\text{drift}} = \begin{cases} w_d * (e^{0.8*\delta} - 1), & \text{if } |\text{drift}| \leq 90^\circ \\ -0.5, & \text{otherwise} \end{cases} \tag{3}$$

where δ is the normalized drift,

$$\delta = \frac{|\text{drift}|}{45^\circ}, \text{drift} \in (-180^\circ, 180^\circ]$$

and w_d is the drift penalty weight (currently -0.005). Drift is defined as the magnitude difference between the

ownship's current heading and the waypoint bearing. The drift penalty exponentially penalizes larger magnitudes of drift, and a drift greater than 90 degrees is severely discouraged, essentially preventing overly large drifts.

3) **Heading Change term**: a penalty applied with the goal of minimizing heading change magnitudes, as shown in Eq. 4:

$$r_{\text{hdg}} = w_h * (e^{0.8*\delta} - 1) \tag{4}$$

Similar to Eq. 3, δ is expressed as the normalized heading change,

$$\delta = \frac{|\Delta\text{hdg}|}{45^\circ}, \Delta\text{hdg} \in (-180^\circ, 180^\circ]$$

Δhdg is defined as the signed difference between the ownship's current heading and its heading at the previous time step. w_h is the heading change penalty weight, which is -0.01. The structure of the heading change penalty mirrors that of the drift penalty, disproportionately punishing larger magnitudes of heading changes.

The drift and heading change penalties are only computed when no intrusion is detected at the current time step (i.e., the ownship is not currently violating separation with an intruder). Because deconfliction is the primary goal, the reward function is designed so that avoiding intrusions takes precedence; minimizing drift and heading change becomes relevant only once this constraint is satisfied. Thus, the drift and heading change penalties are nonzero when the ownship is not currently within the LOS distance of an intruder and zero otherwise.

3. Language-Based Observation Interface

To enable a prompt-based control, we developed a language wrapper that converts structured numerical observations into a templated prompt. The prompt encodes the aircraft state and traffic scenario in natural language, following the consistent format below:

```
You are a decision support system that aids in avoiding collisions between the ownship and
intruders as the ownship navigates to reach a waypoint.  Here is the observation from the
environment:
1.  The intruder distances are [intruder_distance].
2.  The cosines of the bearing angle differences are [cos_difference_pos].
3.  The sines of the bearing angle differences are [sin_difference_pos].
4.  The differences in the x-component of the speed are [x_difference_speed].
5.  The differences in the y-component of the speed are [y_difference_speed].
6.  The distances to each waypoint are [waypoint_distance].
7.  The cosines of the drift angles are [cos_drift].
8.  The sines of the drift angles are [sin_drift].
Using the above information, please provide a decision on whether to turn left or right or
remain unchanged to avoid a collision with the intruder aircraft.

The decision should be one of the following:  [-1 for left, -0.5 for slightly left, 0 for
unchanged, 0.5 for slightly right, 1 for right].
Only provide the decision as output.
```

This interface allows the GPT-2 policy to reason over domain-specific state information in a language form. While the BlueSky environment uses a continuous action space, the initial implementation discretizes it to 5 distinct actions: `[-1, -0.5, 0, 0.5, 1]` which correspond to ownship relative heading change of -45°, -22.5°, 0°, 22.5°, and 45°, respectively.

4. Policy Model and Action Mapping

We employ a pretrained GPT-2 model [14] as a policy network. Given the observation prompt, the model computes logits over vocabulary tokens. Then, an action head maps the outputs of the last hidden state to discretized action tokens: {-1, -0.5, 0, 0.5, 1}. Finally, we extract a categorical distribution over those action tokens (corresponding to directional changes in the ownship's heading), sample from the distribution, and execute the desired action in the environment.

5. Reinforcement Learning and Fine-Tuning

To adapt the language model to the domain, we fine-tune GPT-2 using reinforcement learning, using an approach based on Vanilla Policy Gradient [30]. Specifically, we fine-tune the final output layer of GPT-2 and the action head to better align the token distribution with high-reward actions. Since the loss is computed only on the action tokens (mapped from the last output token's logits), the updates primarily affect the last layers. This approach is computationally efficient and preserves the general linguistic knowledge of the base model while enabling task-specific adaptation. The training loop proceeds as follows:

- The model receives a prompt and samples an action a_t for execution in state s_t.
- The environment returns a new state s_{t+1} post-action execution and a scalar reward r_t.
- Based on the reward, the model is updated using a policy gradient-style loss:

$$\mathcal{L} = -\log \pi(a_t \mid s_t) \cdot r_t \tag{5}$$

where:

- $\mathcal{L}$ is the scalar loss function to be minimized.
- $\pi(a_t \mid s_t)$ is the probability of taking action a_t given the current state s_t, as predicted by the policy π (in our case, GPT-2).
- $\log \pi(a_t \mid s_t)$ is the log-likelihood of the selected action under the current policy.
- r_t is the scalar reward received after taking action a_t.

This setup aligns the model's decision-making with environment feedback, enabling it to learn context-sensitive behaviors without supervised labels.

IV. Results and Discussion

This section presents the performance and effectiveness of the current implementation of our framework on natural language instructions translation, as well as the efficacy of fine-tuning LLMs using RL for autonomous aircraft deconfliction.

A. NL-to-API Translation Setup

The downstream translation agent was implemented under the LangGraph [31] framework with access to a local Ollama server hosting three Nvidia RTX A6000 GPUs; however, only one GPU was used for this study. Implementation followed the details outlined in section III, with the exception of not including the optional node components; instead, a direct edge from the *API Command Selector* to the *Command Parameter Selector* is used for faster evaluations. Evaluated LLMs included Llama3:8B, Qwen2.5-Coder:3B, and Qwen2.5:7B. As a baseline, Qwen2.5:7B was evaluated in single-agent mode, i.e., only a single node was used in LangGraph. To evaluate the LLMs, we created a corpus consisting of domain-specific language derived from BlueSky's trafscript language. Examples of this corpus can be seen in Table 1.

Table 1 Sample of natural language aircraft deconfliction prompts and the resulting BlueSky function invocations consistent with the ingested API.

Natural Language Prompt	Output (BlueSky Function Call)
"Add a Boeing 777 named AC01 going 500 knots at 50 degrees north and 66 degree west heading 97 degrees at 30k ft"	CRE AC01 B777 50.0 -66 97 30000 500
"In order to avoid a conflict between AC01 and AC02 we should add a waypoint for aircraft AC01 at 75 deg west, 10 deg north."	ADDWPT AC01 10.0 -75.0
"I want to add a waypoint to aircraft1 at 10 degrees east and 45 degree north at 32k feet"	ADDWPT aircraft1 45.0 10.0 32000
"Lower the altitude of the colliding aircraft Conflict1 to around 27000 feet?"	ALT Conflict1 27000
"Can you speed this simulation up?"	FF

B. NL-to-API Translation Results

Our NL-to-API Command Translator, as described in III.A, demonstrated a promising approach to processing natural language instructions for execution in the BlueSky environment. The agentic architecture-based method significantly outperformed single LLM approaches, increasing translation accuracy from 43% to 71%, as can be seen in Table 2. Additionally, the choice of LLM used affected the results, with Qwen2.5:7B performing the best in multi-agent mode. Although these results are preliminary, they indicate that multi-agent LLM systems offer promising capabilities for NLP and tool-calling tasks when integrated with external environments.

Table 2 Accuracy comparison for generating BlueSky commands from natural language instructions.

Model	Accuracy
Qwen2.5:7B (Single-Agent)	43%
Qwen2.5:7B (Multi-Agent)	71%
Qwen2.5-Coder:3B (Multi-Agent)	44%
Llama3:8B (Multi-Agent)	50%

C. Deconfliction Setup

The reinforcement learning fine-tuning for the deconfliction agent was conducted on a MacBook Pro equipped with an Apple M2 chip (10-core GPU) using the Metal Performance Shaders (MPS) back end. After training, we evaluated the LLM-based policy in the same environment over 1000 episodes. To assess the effectiveness of language-model-based policies, we compared the performance of a fine-tuned GPT-2 policy with an untrained (zero-shot) GPT-2 baseline, an untrained neural network (essentially a random policy), and a trained neural network. The neural network's architecture is a simple multi-layer perceptron consisting of several sequential linear layers with ReLU activations, ending in a linear output layer whose logits parametrize a categorical distribution over the same discrete actions as described in III.C.4. The neural network and GPT-2 are trained on the same setup, with the reward function detailed in III.C.2, and a lookahead time (for conflict detection) of 300 seconds. Both agents were trained until convergence to the highest success rate possible (100,000 episodes for the neural network, 20,000 episodes for the GPT-2). Furthermore, the TLOS of the intruders in the training setup is randomly selected between [120, 1000] seconds. All agents were evaluated with a lookahead of 600 seconds and a TLOS range of [300, 1000] seconds; the motivation was to train the agents on harsher scenarios so the policy converges faster and behaves less conservatively. Furthermore, to ensure consistency in evaluation, a fixed random seed was used.

D. Deconfliction Results

The results are summarized in Table 3. **Success rate** denotes the percentage of episodes in which the ownship successfully reached the waypoint without any collisions. In the table, $\overline{r_{\text{ep}}}$ denotes the average cumulative reward across all episodes.

Table 3 Comparison of fine-tuned vs. default GPT-2 policy vs random action agent performance. Fine tuning significantly improved safety and goal-reaching behavior.

Agent	Success Rate	Average Reward $\overline{r_{\text{ep}}}$	Standard Deviation ($\sigma(r_{\text{ep}})$)
Untrained Neural Network	64.3%	-2.296	0.278
Untrained GPT-2 LLM	65.9%	-2.828	0.282
RL Trained Neural Network	96.1%	0.578	0.070
RL Fine-Tuned GPT-2 LLM	94.8%	-0.755	0.213

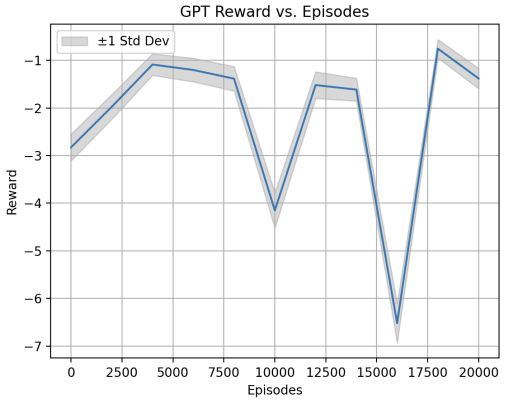
As shown, the reinforcement-learning fine-tuned GPT-2 model demonstrated a substantial improvement in both success rate and reward compared to an untrained GPT-2. Notably, the success rate increased nearly 30%, from 65.9% to

94.8%, while the reward becomes significantly less negative, increasing from approximately -2.83 to -0.76. Furthermore, the success rate of the fine-tuned LLM (94.8%) is comparable to the trained neural network (96.1%), demonstrating that large language models' decision-making in ATC tasks can indeed be enhanced through RL fine-tuning.

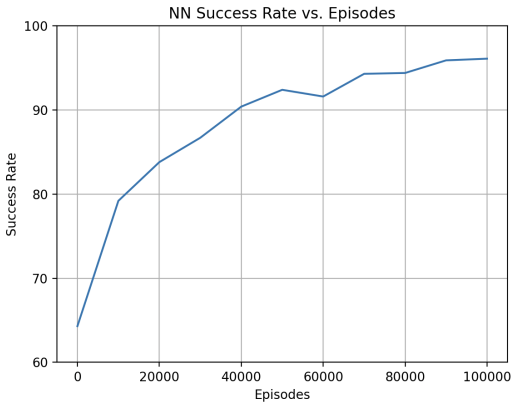
Although the success rates of the trained GPT-2 and neural network are comparable, the neural network exhibited a noticeably higher average reward, meaning that the GPT-2 still induced some unnecessary deviations and heading changes. After more training episodes, the neural network also displayed a significantly smaller standard deviation in reward from episode to episode, while the GPT-2's standard deviation only slightly reduced. These performance differences likely stem from the GPT-2's much larger trainable parameter space (>7 million trainable parameters across the final layer and action head, compared to the neural network's 12,000) as well as the inherent numbers-to-text barrier that the LLM must overcome.

Interestingly, the GPT-2 achieved its peak success rate much earlier in training (after just 18,000 episodes trained), while the neural network converged after roughly 100,000 episodes. However, training beyond 20,000 episodes caused the GPT-2's performance to degrade, suggesting that the model is much more sensitive to extended reinforcement learning updates and may easily be drawn away from the desirable behaviors it acquires.

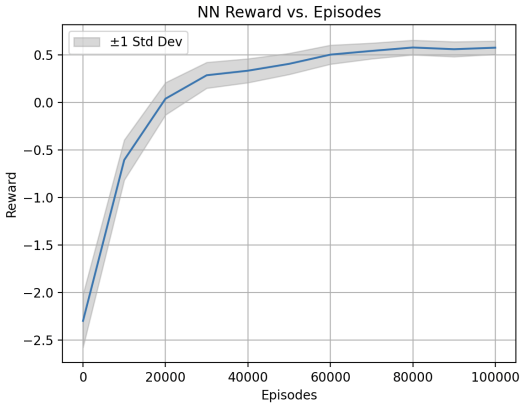
(a) Average Success Rate (GPT-2)



(b) Average Reward (GPT-2)



(c) Average Success Rate (Neural Network)



(d) Average Reward (Neural Network)

Fig. 3 Average evaluation metrics as a function of training progress

Though the GPT-2 reached its highest success rate much earlier than the neural network, its training is significantly noisier. Figure 3 displays the reward and success rate for both the GPT and neural network as a function of the number of episodes trained. The neural network displays a convincing increase in success rate and reward as it is trained for longer. On the other hand, the oscillations in both reward and success rate for the GPT-2 indicate some level of instability when applying reinforcement learning to large language models in complex control settings. Although the

average success rate and reward increase with more training episodes, the trend is not as clear due to sizable fluctuations. However, despite noisy updates, the policy does incrementally learn to make safer and more effective decisions; the GPT-2 still demonstrates a prominent increase in reward and success rate when directly comparing the untrained and trained policies.

Much of the GPT-2's instability can be attributed to the complex nature of the multi-objective task, as well as inherent mismatches between the inputs, outputs, and model architecture. Since the reward function considers many factors (intrusion avoidance, minimal deviation, minimal heading change), significant training time is required for the agent to learn which factors to prioritize. Furthermore, since the GPT-2 must translate numerical state inputs into a text-based embedding space and then (via the action head) map those embeddings back into a discrete numerical action space, small shifts in token-level representations can produce disproportionately large changes in policy behavior. As a result, policy gradients are far more volatile for the GPT-2 compared to the neural network during training and causes the exhibited oscillations.

However, despite this training instability, a large parameter count, and modality mismatch, the GPT-2 is still able to learn an effective and accurate deconfliction policy that performs comparably to a neural network in terms of success rate. This suggests that LLMs, despite modality mismatches and noisy learning dynamics, retain meaningful potential as decision-making agents in complex decision-making settings, motivating further work and investigation into stabilizing RL pipelines and training algorithms for language-based models.

Overall, the fine-tuned GPT-2's high success rate demonstrates that large language models can learn multi-objective deconfliction behavior in complex and dynamic environments, but reducing training noise remains a challenge. The goal of this work is not to demonstrate that embodied LLMs will outperform traditional methods, but rather to illustrate that they are capable decision-making agents when adapted properly. These results highlight three key points: (1) LLMs are capable of learning accurate and realistic deconfliction policies; (2) their ability to interface with natural language offers unique potential for transparent and interpretable interaction with human ATCs; and (3) their performance can be meaningfully improved through reinforcement learning fine-tuning.

V. Conclusion and Future Work

We introduced a novel AI framework, developed around large language models, agentic architecture, and retrieval-augmented generation. Our contribution enables intuitive human-machine interaction with natural language instructions. The framework is designed for seamless integration with different simulation and execution environments by automatically extracting relevant information from API documentation. We presented an aircraft deconfliction case study in Air Traffic Management, where natural language communication is a critical component, but is simultaneously in dire need of decision-support tools to reliably handle the projected increase in operational complexity and air traffic volume.

To this end, this study introduced an LLM-based agent that can effectively resolve aircraft trajectory conflicts, with the potential to act as an assistive tool alongside human ATC operators for the testing and validation of new collaborative human-machine research. We further show that through minimal Reinforcement Learning fine-tuning, multi-agent deconfliction tasks can be successfully solved by significantly smaller language models (such as GPT2) using a fraction of the typical memory and compute resources required by state-of-the-art LLMs.

Future work will focus on expanding the accuracy, stability, and integrability of these LLM agents and frameworks. A key direction is tightly integrating the translation framework with a similar fine-tuned deconfliction LLM agent by constructing a comprehensive LLM pipeline that can process natural language instructions, generate conflict resolution actions, translate those actions into executable commands, and deploy them in real time. The LLM agent's role could also be expanded to include conflict detection, the latter of which was relegated to a separate rule-based method.

Furthermore, future work can go towards improving the accuracy of the translator by utilizing larger LLMs and potentially exploring RL fine-tuning for translation. For the deconfliction agent, next steps include improving the fine-tuning process to be more robust, particularly addressing the sensitivity and instability observed in GPT-2's training states. Sensitivity analysis can be conducted on the parameters of the reward function, state representations, training algorithms, and model parameterizations to gain a comprehensive understanding of the optimal training process to produce a high-quality policy and stable policy evolution during training.

Finally, the LLMs' translation and deconfliction capabilities can be expanded to process more complex instructions, higher-traffic multi-aircraft scenarios, and incorporate richer situational context. A thorough evaluation of a variety of LLM models, RL algorithms and reward functions, and RAG approaches will determine the best-performing configuration for Air Traffic Management environments. Collectively, these investigations will transform LLM incorporation into ATM from proof-of-concept to a reliable component of next-generation ATC decision-support

systems, in which LLM agents act as collaborative teammates that assist human controllers.

References

[1] Airports Council International, I. C. A. O., "Joint ACI World-ICAO Passenger Traffic Report, Trends, and Outlook," *Advisory Bulletins*, 2025. URL https://aci.aero/2025/01/28/joint-aci-world-icao-passenger-traffic-report-trends-and-outlook/.

[2] Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al., "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, Vol. 18, No. 6, 2024, p. 186345.

[3] Jiang, Y., Gupta, A., Zhang, Z., Wang, G., Dou, Y., Chen, Y., Fei-Fei, L., Anandkumar, A., Zhu, Y., and Fan, L., "Vima: General robot manipulation with multimodal prompts," *arXiv preprint arXiv:2210.03094*, Vol. 2, No. 3, 2022, p. 6.

[4] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A., "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.

[5] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al., "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, Vol. 33, 2020, pp. 9459–9474.

[6] Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J., and Wen, J.-R., "Tool learning with large language models: A survey," *Frontiers of Computer Science*, Vol. 19, No. 8, 2025, p. 198343.

[7] Abdulhak, S., Hubbard, W., Gopalakrishnan, K., and Li, M. Z., "CHATATC: Large Language Model-Driven Conversational Agents for Supporting Strategic Air Traffic Flow Management," , No. arXiv:2402.14850, 2024. https://doi.org/10.48550/arXiv.2402.14850, URL http://arxiv.org/abs/2402.14850, arXiv:2402.14850 [cs].

[8] Zhou, W., Wang, J., Zhu, L., Wang, Y., and Ji, Y., "Flight Arrival Scheduling via Large Language Model," *Aerospace*, Vol. 11, No. 10, 2024, p. 813. https://doi.org/10.3390/aerospace11100813.

[9] Vouros, G., Papadopoulos, G., Bastas, A., Cordero, J. M., and Rodrigez, R. R., "Automating the resolution of flight conflicts: Deep reinforcement learning in service of air traffic controllers," , No. arXiv:2206.07403, 2022. https://doi.org/10.48550/arXiv.2206.07403, URL http://arxiv.org/abs/2206.07403, arXiv:2206.07403 [cs].

[10] Yadav, S., "AeroQuery RAG and LLM for Aerospace Query in Designs, Development, Standards, Certifications," *2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, IEEE, 2024, pp. 1–6.

[11] Grigorev, A., Saleh, K., Ou, Y., and Mihăiţă, A.-S., "Enhancing Traffic Incident Management with Large Language Models: A Hybrid Machine Learning Approach for Severity Classification," *International Journal of Intelligent Transportation Systems Research*, Vol. 23, No. 1, 2025, pp. 259–280.

[12] Nielsen, D., Clarke, S. S., and Kalyanam, K. M., "Towards an aviation large language model by fine-tuning and evaluating transformers," *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, IEEE, 2024, pp. 1–5.

[13] Hoekstra, J. M., and Ellerbroek, J., "Bluesky ATC simulator project: an open data and open source approach," *Proceedings of the 7th international conference on research in air transportation*, Vol. 131, FAA/Eurocontrol Washington, DC, USA, 2016, p. 132.

[14] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al., "Language models are unsupervised multitask learners," *OpenAI blog*, Vol. 1, No. 8, 2019, p. 9.

[15] Singh, U., Bhattacharya, M., and Padhi, R., "State-of-the-Art Natural Language Processing for Aviation: A Review," *Authorea Preprints*, 2025.

[16] Wang, L., Chou, J., Tien, A., Zhou, X., and Baumgartner, D., "AviationGPT: A large language model for the aviation domain," *AIAA AVIATION FORUM AND ASCEND 2024*, 2024, p. 4250.

[17] Yang, J., Xiang, X., and Chen, X., "A Retrieval-Augmented Generation-Based Method for Aviation Accident Data Analysis," *2024 4th International Conference on Artificial Intelligence, Robotics, and Communication (ICAIRC)*, IEEE, 2024, pp. 868–874.

[18] Shen, Z., "Llm with tools: A survey," *arXiv preprint arXiv:2409.18807*, 2024.

[19] Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., Zeng, Z., Zhou, X., Huang, Y., Xiao, C., Han, C., Fung, Y. R., Su, Y., Wang, H., Qian, C., Tian, R., Zhu, K., Liang, S., Shen, X., Xu, B., Zhang, Z., Ye, Y., Li, B., Tang, Z., Yi, J., Zhu, Y., Dai, Z., Yan, L., Cong, X., Lu, Y., Zhao, W., Huang, Y., Yan, J., Han, X., Sun, X., Li, D., Phang, J., Yang, C., Wu, T., Ji, H., Li, G., Liu, Z., and Sun, M., "Tool Learning with Foundation Models," *ACM Computing Surveys*, Vol. 57, No. 4, 2025, p. 1–40. https://doi.org/10.1145/3704435.

[20] Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J., and Wen, J.-R., "Tool Learning with Large Language Models: A Survey," *Frontiers of Computer Science*, Vol. 19, No. 8, 2025, p. 198343. https://doi.org/10.1007/s11704-024-40678-2, arXiv:2405.17935 [cs].

[21] Shi, Z., Gao, S., Yan, L., Feng, Y., Chen, X., Chen, Z., Yin, D., Verberne, S., and Ren, Z., "Tool Learning in the Wild: Empowering Language Models as Automatic Tool Agents," *Proceedings of the ACM on Web Conference 2025*, ACM, Sydney NSW Australia, 2025, p. 2222–2237. https://doi.org/10.1145/3696410.3714825, URL https://dl.acm.org/doi/10.1145/3696410.3714825.

[22] Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Huang, M., Duan, N., and Chen, W., "ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving," , No. arXiv:2309.17452, 2024. https://doi.org/10.48550/arXiv.2309.17452, URL http://arxiv.org/abs/2309.17452, arXiv:2309.17452 [cs].

[23] Di Sipio, C., Rubei, R., Di Rocco, J., Di Ruscio, D., and Iovino, L., "On the use of LLMs to support the development of domain-specific modeling languages," *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ACM, Linz Austria, 2024, p. 596–601. https://doi.org/10.1145/3652620.3687808, URL https://dl.acm.org/doi/10.1145/3652620.3687808.

[24] Andriuškevičius, J., and Sun, J., "Automatic Control With Human-Like Reasoning: Exploring Language Model Embodied Air Traffic Agents," *arXiv preprint arXiv:2409.09717*, 2024.

[25] Tabrizian, A., Gupta, P., Taye, A., Jones, J., Thompson, E., Chen, S., Bonin, T., Eberle, D., and Wei, P., "Using Large Language Models to Automate Flight Planning Under Wind Hazards," *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, IEEE, 2024, pp. 1–8.

[26] Brittain, M. W., Yang, X., and Wei, P., "Autonomous Separation Assurance with Deep Multi-Agent Reinforcement Learning," *Journal of Aerospace Information Systems*, Vol. 18, No. 12, 2021, p. 890–905. https://doi.org/10.2514/1.I010973.

[27] Brittain, M. W., Alvarez, L. E., and Breeden, K., "Improving Autonomous Separation Assurance through Distributed Reinforcement Learning with Attention Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, No. 2121, 2024, p. 22857–22863. https://doi.org/10.1609/aaai.v38i21.30321.

[28] Chen, W., Koenig, S., and Dilkina, B., "Why solving multi-agent path finding with large language model has not succeeded yet," *arXiv preprint arXiv:2401.03630*, 2024.

[29] Groot, D., Leto, G., Vlaskin, A., Moëc, A., and Ellerbroek, J., "BlueSky-Gym: Reinforcement Learning Environments for Air Traffic Applications," 2024.

[30] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M., "Deterministic policy gradient algorithms," *International conference on machine learning*, Pmlr, 2014, pp. 387–395.

[31] Mavroudis, V., "LangChain v0. 3," *Preprints*, 2024.

Appendix

NL-to-API Command Translator Prompts:

Command Verifier

```
f"Determine if the user gave you all the information you need to "+\
          f"implement [{cmd}] as per the following rules, examples: {context} \n\n" +\
          f"in the following user request: {state.user_request}. \n\n " +\
          "YES if you have ALL the required parameters to implement it, " +\
          "and NO if you're missing even a single value, like if the user did not "+\
          "provide a custom acid/speed or some other required parameter. " +\
          "Your response should be one word, either YES or NO. "
```

API Command Selector

```
f'{content} \n\n ' +\
       f'The user wants to implement a command from the above list of commands'+\
       f' to do the following operation: \n {request}. \n\n ' +\
       f'Select one command from the list of commands given ' +\
       'and return it as \\boxed{command} ' +\
       f'that the user should use to implement what they outlined. '+\
       'For example, if the user wants to add an aircraft then you should'+\
       ' return \\boxed{CRE} which is the command to create a new aircraft.'
```

Command Parameter Selector/Validator

```
 f'{context} \n\n ' +\
       f'The user wants to implement the command {cmd} as per the given usage criterion'+\
       f' to do the following operation: \n {request}. \n\n ' +\
       'Follow the syntax provided to output a valid instance of the command as \\boxed{
           usage of command}' +\
       'You have to produce an instance of the command, complete will all filled in values
           that may be required according to the specification given.'
```