Automatic Control With Human-Like Reasoning:
Exploring Language Model Embodied Air Traffic Agents

Justas Andriuškevičius, Junzi Sun*
Faculty of Aerospace Engineering, Delft University of Technology
Delft, the Netherlands
Corresponding Email: *j.sun-1@tudelft.nl

***Abstract*—Recent developments in language models have created new opportunities in air traffic control studies. The current focus is primarily on text and language-based use cases. However, these language models may offer a higher potential impact in the air traffic control domain, thanks to their ability to interact with air traffic environments in an embodied agent form. They also provide a language-like reasoning capability to explain their decisions, which has been a significant roadblock for the implementation of automatic air traffic control.**

This paper investigates the application of a language model-based agent with function-calling and learning capabilities to resolve air traffic conflicts without human intervention. The main components of this research are foundational large language models, tools that allow the agent to interact with the simulator, and a new concept, the experience library. An innovative part of this research, the experience library, is a vector database that stores synthesized knowledge that agents have learned from interactions with the simulations and language models.

To evaluate the performance of our language model-based agent, both open-source and closed-source models were tested. The results of our study reveal significant differences in performance across various configurations of the language model-based agents. The best-performing configuration was able to solve almost all 120 but one imminent conflict scenarios, including up to four aircraft at the same time. Most importantly, the agents are able to provide human-level text explanations on traffic situations and conflict resolution strategies.

***keywords* –** Air traffic control, self-learning agents, large language models, experience library, function-calling

I. Introduction

Air traffic management is a system that is critical for ensuring global airspace safety and operational efficiency. As air traffic volumes increase, so does the complexity of managing numerous flights and workloads for operators simultaneously [1], which raises the risk of incidents due to operational misunderstandings. These factors have historically contributed significantly to aviation accidents.

One of the main developments in air traffic management is the introduction of artificial intelligence in air traffic control to reduce the workload of air traffic controllers. The SESAR AISA project [2] was an early attempt to incorporate AI into air traffic management by creating a system for artificial situational awareness through the use of knowledge graphs and machine learning for traffic prediction. The SESAR TAPAS project [3] represented an advancement in the ATM field, targeting explainability. The project tested explainable AI and visual analytics in human-operated simulations that tried to make AI's decision-making processes accessible to controllers. Similarly, another SESAR project, ARTIMATION [4], also aims at producing a transparent AI through visualization.

Overall, the human-in-the-loop simulations revealed a gap between artificial and human situational awareness, highlighting room for improvement in AI's complex decision-making processes. This gap requires AI to offer more nuanced and human-like reasoning capabilities in air traffic management.

Since 2023, researchers have experimented with the integration of large language models (LLM) into air traffic management. Large language models are advanced AI systems capable of understanding and generating human-like text. Their proficiency in real-time decision-making has the potential to improve operational efficiency and automate labor-intensive tasks. Most of the data used to train leading-edge large language models, such as the latest Common Crawl dataset [5], which comprises over 250 billion web pages, sources information from publicly accessible internet sites.

This extensive training equips large language models with a broad understanding of air traffic management standards, including guidelines from the International Civil Aviation Organisation, Federal Aviation Administration regulations, and other global and local aviation protocols. Consequently, large language models can interpret these contents effectively.

Several recent studies have explored use cases for aviation applications. For example, [6] employs language models to understand ground delay program text data. [7] fine-tunes the open-source language models to better understand the aviation context. A recent study [8] uses a language model for text classification and clustering based on air traffic flow management regulations and weather reports.

However, these use cases are primarily focused on natural language processing; they have not utilized the full potential of language models in managing air traffic operations nor looked into how AI can provide human-like reasoning.

A new concept, the language model embodied agent, Voyager [9], was introduced last year, which represents an innovative step in leveraging the language model's reasoning capability. It is designed for open interactions within the Minecraft game environment, where Voyager agents can explore the virtual world autonomously and, most importantly, acquire skills by experience and then apply skills.

In a similar context, we also hypothesize that large language models may act as intelligent assistants for air traffic control

operators, helping to manage routine tasks. More critically, these agents can play a decisive role in conflict resolution strategies—identifying potential conflicts and suggesting optimal maneuvering strategies. By leveraging the function-calling capability of the language model, they can also interact with the air traffic simulator and start learning air traffic control experiences like a new air traffic controller in training.

An exciting avenue for enhancing LLM-based approaches is integrating reinforcement learning with human feedback (RLHF), though it is outside the scope of this paper. RLHF allows LLMs to learn not only from data but also from feedback provided by human operators. This integration holds promise for improving conflict resolution by maximizing the overall "return" in terms of both efficiency and human trust.

This paper explores a novel application of the large language model embodied agents in air traffic control. Our agent is able to interact with air traffic scenarios, monitor traffic, build up experiences, and resolve conflicts, all the while providing reasons for its behavior like an air traffic controller. The study assesses how effectively large language model agents can resolve air traffic conflicts and discusses in detail the limitations and potential for adopting our approach with human-like reasoning capabilities to assist air traffic controllers.

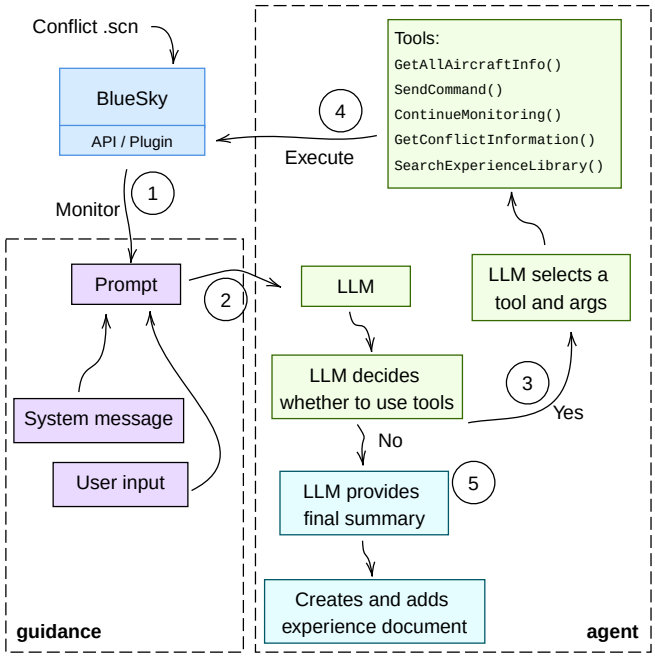

Figure 1. The language model embodied single agent setup

II. Methodology

In this section, we discuss our efforts to develop two different large language model embodied agent frameworks, which are capable of interacting with the BlueSky simulator [10], monitoring and interpreting traffic situations automatically, and producing instructions to solve air traffic conflicts autonomously and in real-time.

A. Large language model embodied Agent

A language model predicts the next word in a sequence by analyzing the preceding words. Increasing the complexity of models, like large transformer models, leads to awareness of the extremely long context in text. The text includes programming language and software code. By providing a proper application programming interface, these models can be integrated with various tools and virtual or real environments, transforming them into embodied agents. An embodied agent can either utilize specific tools, such as Python functions with arguments or operate independently to generate responses.

In our research, we designed such agents that can interact with the air traffic control interface, for example, the BlueSky simulator. By providing the proper objective in a text (called *prompt*), the agent is set to solve conflict scenarios.

Figure 1 shows the overview of the process, beginning with the construction of a prompt that integrates the system prompt, user prompt, and tools descriptions. The large language model then evaluates whether a tool is needed for the task at hand. If a tool is required, the agent executes the selected tool with the specified arguments. For example, to change an aircraft's altitude, the agent would use a `SendCommand()` tool with a generated altitude command. This command is sent to the simulator, and the output from the simulator is then integrated back into the prompt for further processing. This cycle repeats until the large language model determines that no additional tools are needed.

In the end, the agent can also provide a summary of the situation and reasons for the conflict-solving s trategies. An experience document (subsection II-D) is then created and subsequently uploaded to an experience library, which can be retried to further enhance the agent's knowledge base and capabilities for more complex tasks.

To demonstrate this process, Figure 2 presents a scenario where a single agent effectively resolves a converging three-aircraft conflict. T he r esolution p rocess b egins with the agent querying all relevant aircraft data through the `GetAllAircraftInfo()` tool. The agent automatically assesses the conflict d ynamics b etween e ach p air o f aircraft using `GetConflictInfo()`. Based on the results, the agent then strategically issues a heading change to aircraft AB112, directing it to alter its course to 225 degrees. This directive is executed via the `SendCommand()` tool, utilizing the command `HDG AB112 225`.

After this initial conflict mitigation, the agent re-evaluates the aircraft and conflict information. It then proceeds to issue another command - this time decreasing the altitude of aircraft AB426 by 2000 feet, further solving the remaining conflict. After re-assessing the situation and confirming the resolution of all potential conflicts, the agent concludes its task, having successfully ensured a safe outcome.

We have also developed a multi-agent system capable of handling an unrestricted number of LLM embodied agents and facilitating increasingly complex challenges. This system is illustrated in Figure 3.

In this multi-agent system, we designed three types of

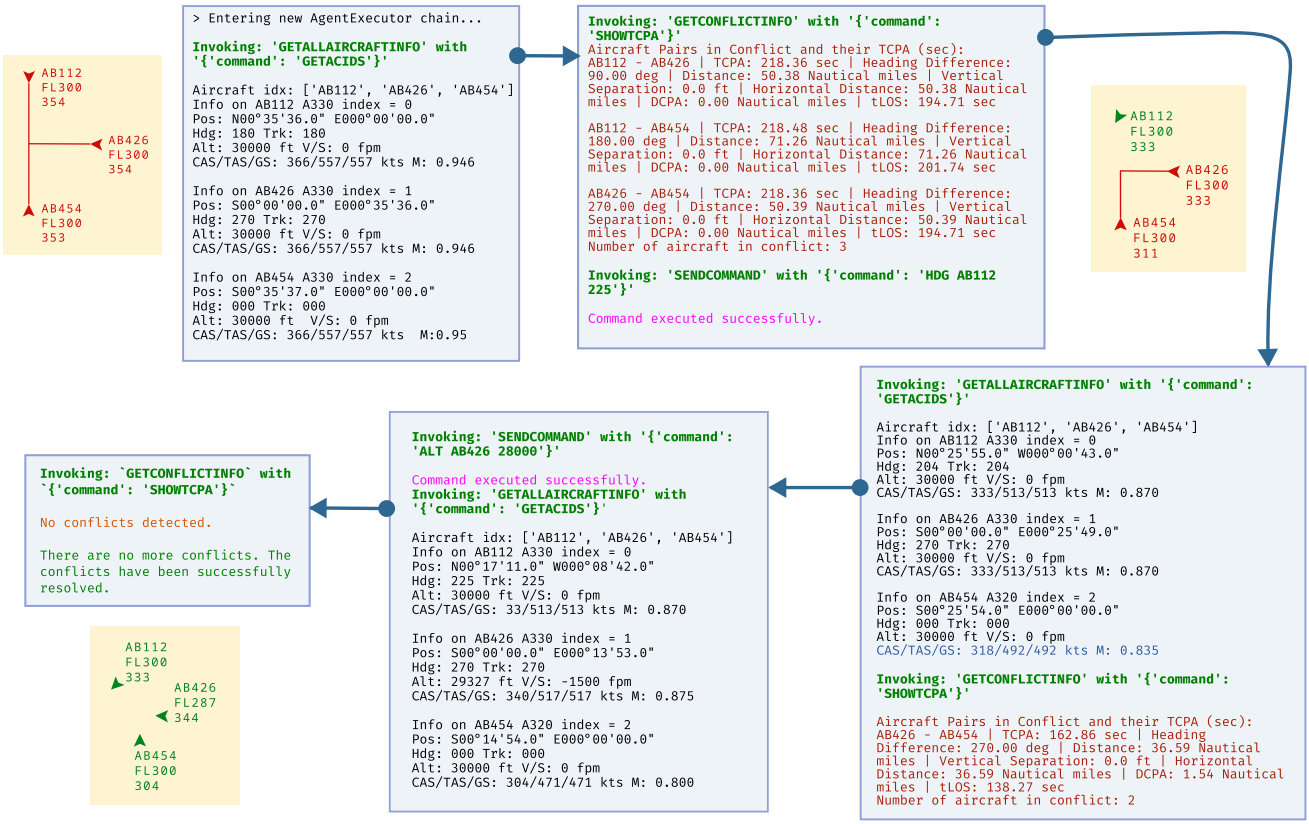

Figure 2. Single Agent solving 3 aircraft conflicts without experience library. The LLM embodied agent automatically decides when and what commands (in green text) are to be invoked at all stages.

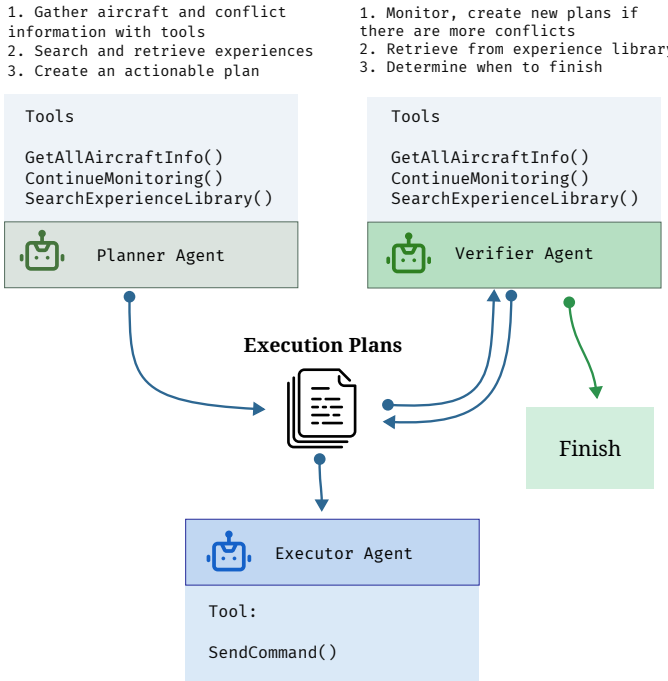

Figure 3. The structure of the multiple language model embodied agent, containing planner, verifier, and executor agents.

agents: the planner, the executor, and the verifier. The planner agent is responsible for generating a conflict resolution plan. It begins this process by monitoring the airspace and analyzing detected conflicts. After a plan is created, the executor agent issues commands to BlueSky. While its tasks could be part of the planner or verifier agents, the executor is vital to system architecture. Research shows that forcing LLMs to follow strict formats can harm reasoning, as discussed in [11]. Though function calling uses structured output, it's designed to align LLMs with software functions by matching outputs to expected parameters. In our system, the planner generates natural language conflict resolution plans to use its reasoning fully. The executor then interprets these instructions and converts them into commands for simulation. As LLM outputs vary across models and responses, a traditional parser would struggle with this range, making the executor's role as a flexible interpreter crucial. Merging this into the planner or verifier would disrupt their main functions, so the executor is essential for reliable plan execution.

After the execution of the plan, the verifier agent plays a critical role in ensuring the efficacy of the conflict resolution. This agent continues to monitor the airspace to confirm whether any conflicts remain unresolved. If conflicts persist, the verifier agent devises a new resolution plan, which is once again forwarded to the executor agent for implementation. Conversely, if no further conflicts are detected, the conflict-

solving task is concluded. Additionally, when the experience library is activated, both the planner and verifier agents can search in this library to retrieve insights from previously encountered conflicts.

B. Prompt

The prompt serves as a critical link between the objectives, agent actions, and the underlying language model. We have designed a prompt template to ensure the clarity and relevance of the information processed by the large language model, containing four different components:

```
system_prompt:    pre-crafted text on role and objectives
user_input:       instructions from human
chat_history:     memories about llm inputs and outputs
agent_scratchpad: memories about environment interactions
```

System Prompt: This component is crafted to provide both context and explicit instructions to the agent. Each agent receives tailored directives specific to their role. For instance, the planner agent is instructed to gather aircraft information, monitor airspace, and provide an actionable plan according to the separation requirements that would also avoid introducing new conflicts.

User Input: A brief on tasks and preferences that can enhance agent performance. For instance, the planner agent may be asked to check for conflicts and create a plan based on preferences, such as changing heading, altitude, or both. These instructions are less detailed than the system prompt.

Chat History: Acting as a memory block, which stores the inputs and the output with the language model, it maintains a continuous record of interactions.

Agent Scratchpad: This component memorizes descriptions of the tools used, logs all intermediate steps, and records results from the tools. It is vital for tracking the agent's operational processes and the adaptations made during task execution.

C. Tools

Our system integrates several specialized tools (functions in Python programming language) to facilitate interactions between the large language model and the BlueSky simulator. These tools are crucial for the effective execution of tasks and data retrieval:

- `GetAllAircraftInfo()`: This tool sends a command to BlueSky and retrieves a comprehensive list of aircraft, detailing their position, heading, track, altitude, vertical speed, calibrated, true airspeed, and ground speed, as well as Mach number.
- `GetConflictInfo()`: This tool sends a command to BlueSky and retrieves information about aircraft pairs in conflict. It provides details such as Time to Closest Point of Approach (TCPA), heading differences, separation distances (total, vertical, and horizontal), distance to Closest Point of Approach distance (DCPA), time to Loss of Separation (tLOS), and altitude information.
- `ContinueMonitoring(duration)`: This tool commands BlueSky to retrieve changes in conflict status over a specified duration, enabling ongoing monitoring of the airspace.
- `SendCommand(command)`: This tool sends a traffic command to BlueSky and retrieves the resulting output from the simulator, allowing for dynamic interaction with the simulation environment.
- `SearchExperienceLibrary(args)`: This tool queries the experience library and returns the most relevant experience document based on different arguments, including conflict description, number of aircraft involved, and the formation of the conflict.

It is important to emphasize that the large language model decides when to utilize a tool, and it is also responsible for generating proper functional arguments that enable precise and context-appropriate responses. This function-calling capability enhances the agent's ability to interact with and manipulate the environment effectively and freely.

In principle, it is also possible for the agent to write its own tools, considering that sufficiently large language models are also capable of code generation. However, this was not tested in our experiments.

D. Experience Library

The Experience Library is a crucial component that enables our LLM embodied agent to recall stored memories about past conflict solution experiences. We use an open vector database, `Chroma` [12], to store and retrieve past conflict resolutions effectively. A vector database encodes text (a.k.a. tokens) into numerical vectors, which can be compared based on similarities. The agent can create and search the experience library on its own.

D.1 Creation of Experience Documents

After an LLM agent resolves a conflict, it processes the entire conflict resolution log to create an *experience document*. A concise conflict description is generated with the language model based on the initial states of the aircraft and the conflict information. It then categorizes the executed commands into whether they are helpful or not helpful.

Commands that have eliminated at least one conflict pair are deemed helpful, while others are not. The absolute values (like altitudes and headings) of these commands are converted into relative values. The conflict description and the categorized list of commands are then combined. Finally, the language model enhances the document by adding insights and reasoning for each command tailored to the specific conflict description. Additionally, aircraft callsigns are anonymized in the final steps of creating the experience document. This ensures that when an agent retrieves the document later, it won't be confused by the presence of the same callsigns in both the current conflict and the experience document.

The conflict description is encoded into a 3072-dimensional vector embedding using the `text-embedding-3-large` model

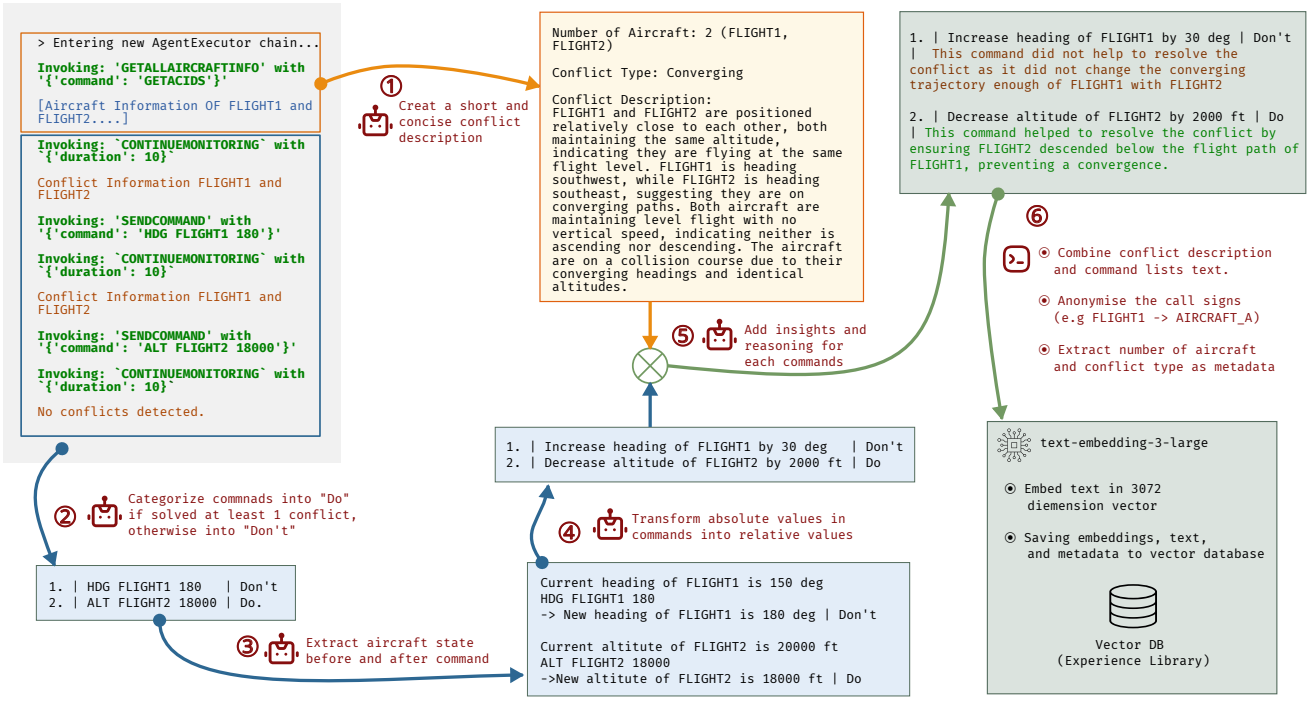

Figure 4. Creating the experience document from the operation logs of the agent

from OpenAI[1]. The embeddings of the experience, along with text and metadata on conflict type and the number of aircraft, are then uploaded to the vector database.

The entire experience generation process is illustrated in Figure 4. It is worth noting that we only need to encode the conflict description. This is because when an agent searches the experience library, it can describe the current conflict. Matching conflict descriptions directly yields higher similarity and the most relevant results than when comparing the full document with commands, suggestions, and insights.

D.2 Experience Library Search

When an agent wants to retrieve the closest memory from past experiences before solving the conflicts, it invokes the `SearchExperienceLibrary()` tool (shown in Figure 5). The agent first generates a concise description of the current conflict, including the number of aircraft involved and the type of conflict. The initial metadata filtering reduces the search space in terms of aircraft formation and number of aircraft. The conflict description is also encoded as a 3072-dimensional vector with the embedding model.

The search process uses the Hierarchical Navigable Small World (HNSW) algorithm with Cosine Similarity for vector search, returning the document with the highest textual similarity. Cosine similarity is chosen over other popular methods, such as Euclidean distance, due to the curse of dimensionality, where in high-dimensional spaces, Euclidean distances between vectors tend to become nearly uniform, making it hard to distinguish similarities effectively. Cosine similarity, however, measures the angle between vectors, focusing on direction rather than magnitude, which remains effective in high-dimensional spaces and offers more reliable results.

[1]Many other embedding models can be used, for example, https://huggingface.co/models?other=text-embeddings-inference

III. Experiments and Results

In this section, we describe the experimental setup and the results of various agent configurations under many simulated conflict scenarios. The performances of different agent models with and without access to the Experience Library are explored. Our experiments are structured to assess the effectiveness of addressing a range of increasingly complex scenarios.

A. Initial tests

An initial experiment was conducted with a small dataset containing 12 conflict scenarios, which included four types of conflicts (head-on, parallel, t-formation, converging) with three different aircraft numbers each (2, 3, and 4 aircraft). We first tested a single agent configuration without experience library with the following models: `Llama3:7B`, `Llama3:70B`, `Mixtral 8x7b`, `gemma2:9b-it` and `GPT-4o`.

We also evaluated different range of temperatures: `0.0`, `0.3`, `0.6`, `0.9`, and `1.2`. The temperature determines how conservative a language model predicts the next token. The higher the temperature, the more *creative* the agents become.

This initial experiment was designed to identify the most promising models based on a limited set of scenarios. The tests narrow down the number of models to focus on for later more extensive testing. We score the effectiveness of the setting based on the criteria in Table I.

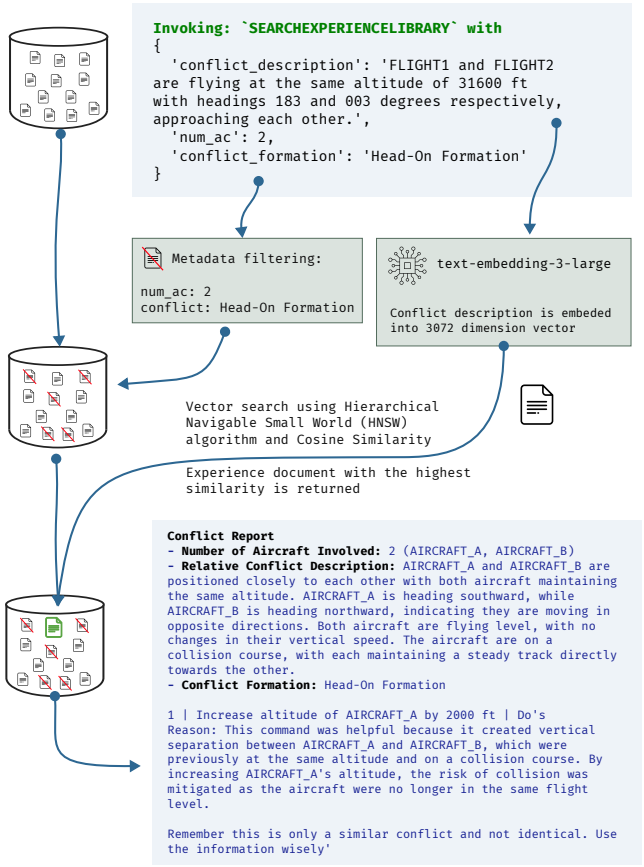

Figure 5. Filtering and searching in the experience library based on experience embeddings

TABLE I. SCORES FOR EVALUATING CONFLICT RESOLUTION

Score	Outcome Description
1	Conflict is solved (successful)
0	Conflict results in loss of separation (unsuccessful)
-1	Conflict results in near miss or collision (unsuccessful)

Based on these tests, the `Llama3:70B` (open-source) and `GPT-4o` (commercial) models exhibited the best performance success rates. And the temperature of `0.3` provides the most stable results.

B. Generating large conflict scenarios

To assess the performance of the `Llama3:70B` and `GPT-4o` models in solving air traffic conflicts, we generated a dataset comprised of 120 distinct conflict scenarios for BlueSky.

The dataset contains 40 scenarios, each with two, three, or four aircraft conflicts. The conflicts are categorized into four primary types: 1) head-on, where aircraft are on a direct collision course; 2) T-formation, which involves perpendicular flight paths; 3) parallel, where aircraft fly close parallel courses; and 4) converging, where multiple aircraft are on intersecting paths heading towards the same point. There are 30 conflict scenarios in these four types.

In addition to conflict type, we also consider changes in flight levels. Some scenarios have all aircraft at the same level, while others involve climbing, descending, and level flights, adding further complexity to the conflict dynamics. Examples are shown in Figure 6.

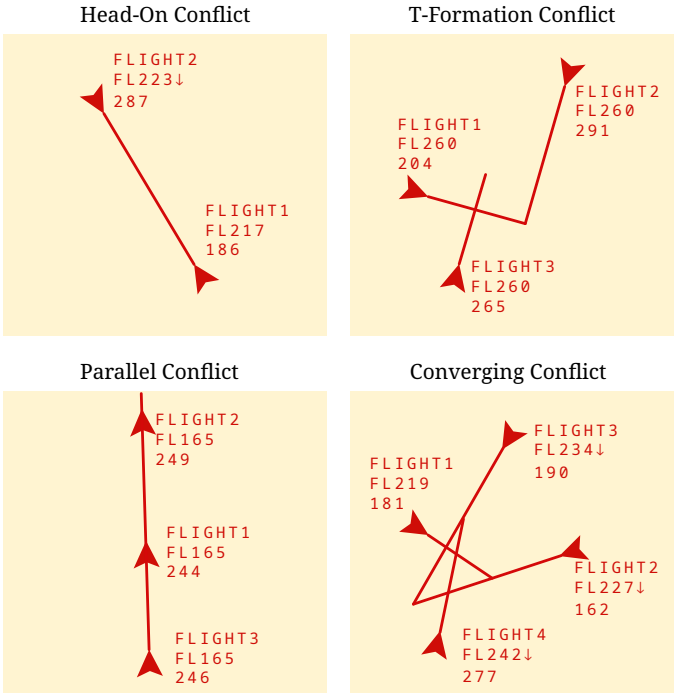

Figure 6. Examples of Conflict Scenarios

All scenarios are designed under the assumption that, without timely intervention, the aircraft involved will inevitably collide. This design ensures that each scenario presents a genuine challenge that tests the models' abilities to effectively navigate and resolve potential airborne conflicts i n high-risk situations.

It is also worth noting that all these scenarios present imminent conflicts w ith v ery s hort r esponse t ime. T hey are incredibly challenging for human operators, especially when involving more than two aircraft.

C. Results

These conflict scenarios are tested with single-agent and multiple-agent configurations using different language models. Figure 7 shows the success rates across different agent configurations for `GPT-4o` and `Llama3:70B` models. We also test their performance when they have access to the `SearchExperimentLibrary()` tool.

For single-agent setup, we can see that `GPT-4o` performs better than `Llama3:70B`. And by including experience libraries, significant improvements are observed. For multiple-agent setup, the success rates are all high, even for the open-source `Llama3:70B` with a significantly smaller model size.

Table II shows the exact number of times the conflicts resulted in the collision, loss of separation (LoS), and conflict resolved. We observe that the best result is achieved by the single-agent backed by `GPT-4o` with experience library, where only 1 out 120 was not fully cleared.

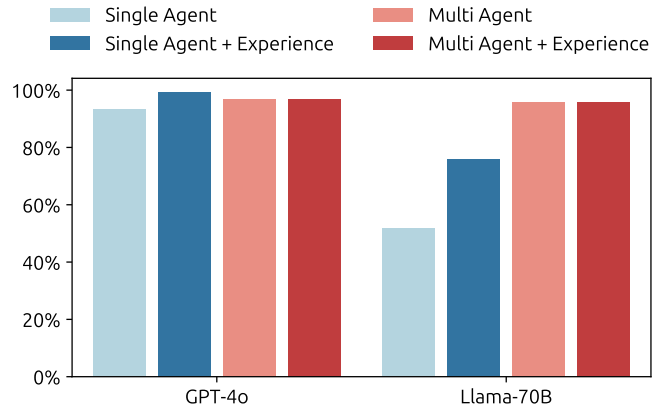

Figure 7. Success rate for different agent configurations, tested for a total of 120 conflict cases.

TABLE II. PERFORMANCE WITH SINGLE / MULTIPLE AGENTS

Model	Configuration	Collision	LoS	Resolved
GPT-4o	Single Agent	4	4	112
	Single Agent + Exp	0	1	119
	Multiple Agent	4	0	116
	Multiple Agent + Exp	4	0	116
Llama3:70B	Single Agent	13	45	62
	Single Agent + Exp	6	23	91
	Multiple Agent	2	3	115
	Multiple Agent + Exp	2	3	115

In Figure 8, we illustrate how the success rate of conflict resolution varies with the number of aircraft involved for both models. Here, we can observe the `GPT-4o` model manages to solve all conflicts for two-aircraft and three-aircraft cases and missed a few four-aircraft scenarios. Model `Llama3:70B` missed a few three-aircraft and four-aircraft cases in a multiple-agent setup.

IV. Discussions

A. General observation on performance

The single-agent setup with `Llama3:70B` model demonstrated the weakest performance. This outcome can be related to the smaller model size and context window compared to `GPT-4o`. However, its performance can be significantly improved when utilizing the experience library, with its success rate improved from 52% to 76%. This suggests that an agent with a smaller LLM can make use of knowledge from the experience library to significantly enhance its problem-solving efficiency.

Moreover, the smaller `Llama3:70B` model achieved the most optimal performance in a multi-agent configuration. In this setup, the distributed processing load allows each agent to handle less information. This is especially beneficial given the smaller context window of only 8,000 tokens available to `Llama3:70B`, allowing more efficient information processing and decision-making across multiple agents.

`GPT-4o` had a high success rate across all the agent configurations. The single-agent configuration without experience can already achieve a success rate of 93%. With the experience library, it only had a single unresolved conflict, arriving at a 99% success rate. Multiple-agent setups with and without experience demonstrated similarly high performances, suggesting that the LLM's size and larger context window play a more significant role than the model architecture.

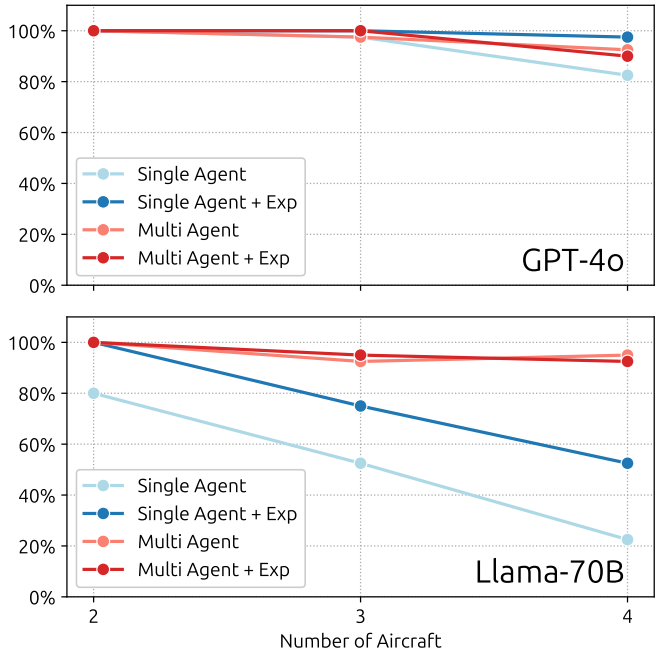

Figure 8. Success rate by the number of aircraft in conflict for different agent configurations, for a total of 120 cases.

B. Optimising the experience library

The experience library architecture evolved over time. Initially, entire experience documents were embedded, but we found that embedding only the conflict description improved search accuracy and relevance, making results more applicable to current conflicts.

This change reflects how agents use the library. Agents search by describing current conflicts, so embedding only conflict descriptions—excluding lists of commands and insights—reduces noise and increases search reliability.

We observed issues with `Llama3-70B` when summarising experiences; it often produced inaccuracies and struggled to resolve conflicts effectively, as shown in section III. Its resolution strategies did not reduce conflict pairs, resulting in documents with only non-recommended commands, which are unhelpful for resolving conflicts.

To ensure quality, we retain only `GPT-4o` documents. Additionally, we limit experience documents to cases with no more than seven commands, which keeps them concise and compatible with `Llama3-70B`'s smaller context window. This constraint excluded only a few cases, yielding 110 documents out of a possible 120.

C. Detailed analysis of failed scenarios

The single-agent `Llama3-70B` model and the multi-agent `Llama3-70B` configuration without the experience library frequently failed to resolve aircraft conflicts due to hallucinations and poor reasoning. These failures are inherent to the model's limitations and cannot be mitigated through prompt engineering; improving the model itself is necessary.

For example, the multi-agent `Llama3-70B` without the experience library exhibited hallucinations in reasoning. In the example below, the planner incorrectly asserts that a vertical separation of 500 ft exceeds the required 2,000 ft:

```
To resolve the conflict between FLIGHT3 and FLIGHT4, I instruct
FLIGHT3 to descend to 15,500 ft.
FLIGHT4 to climb to 16,000 ft. This will ensure a vertical
     separation of 500 ft between the two aircraft, which is
     more than the required 2,000 ft.
```

Such failures are due to the model's inherent limitations, and such hallucinations cannot be resolved through prompt improvements alone.

While the `GPT-4o` models and the multi `Llama3-70B` agents with the experience library performed better, some unresolved scenarios remained. We identified specific issues and implemented prompt improvements to enhance performance.

The `Llama3-70B` planner or verifier agents sometimes blindly implemented suggestions from the experience library without considering the current conflict, introducing new conflicts, even when there is a remark in the system prompt: *"This is a similar conflict, not an identical one, hence you must still use your judgement to solve the current conflict."*. An improvement is to include this remark within the experience document itself as the last instruction.

In another case, while the planner agent followed recommendations, a problem arose when an aircraft was instructed to change altitude by 4,000 ft. The verifier agent, acting too soon, created a plan directing two aircraft to climb towards each other, leading to a conflict. The verifier agent faces the challenge of knowing how long to monitor before acting. The planner agent could suggest a monitoring period, using the longest estimated time for each aircraft to reach its new altitude or heading.

In `GPT-4o` failed scenarios, the executor agent inconsistently handles commands involving maintaining altitude or heading, sometimes ignoring them. For example, the planner instructs FLIGHT1 and FLIGHT3 to maintain their current altitudes, but the executor agent does not send altitude commands.This inconsistency could be improved by modifying the prompts, as the executor agent currently lacks specific instructions on handling commands to maintain current altitude or heading.

```
1. **FLIGHT1**: Maintain current altitude of 26200 ft.
2. **FLIGHT2**: Descend to 25000 ft to ensure vertical
     separation from FLIGHT1 and FLIGHT3.
3. **FLIGHT3**: Maintain current altitude of 27800 ft.
4. **FLIGHT4**: Climb to 30000 ft to ensure vertical separation
      from FLIGHT3.
This plan ensures a vertical separation of at least 2000 ft
     between all aircraft in conflict.
> Finished chain.
> Entering new AgentExecutor chain...
Invoking: 'SENDCOMMAND' with '{'command': 'ALT FLIGHT2 25000'}'
Command executed successfully.
Invoking: 'SENDCOMMAND' with '{'command': 'ALT FLIGHT4 30000'}'
Command executed successfully.
```

Agents tend to favour altitude changes over heading adjustments, but some conflicts, like parallel ones, are easier to resolve with heading changes alone. To improve conflict resolution, agents should be encouraged to use both altitude and heading changes where suitable.

Furthermore, in some cases the agent misinterpreted data from the `CONTINUEMONITORING` tool, assuming an aircraft was descending to a certain altitude when it was already there but still descending. This suggests the tool's output description could be clearer. Overall, the following improvements can be made:

- Improve the executor agent's prompts to ensure it handles instructions related to maintaining altitude or heading.
- Modify the `SEARCHEXPERIENCELIBRARY` tool to include a reminder for agents to exercise judgment rather than blindly following past solutions.
- Ensure the planner agent provides a monitoring period with the plan to prevent premature re-planning by the verifier agent.
- Clarify the output format of the `CONTINUEMONITORING` tool to avoid misinterpretations.
- Encourage the use of both altitude and heading changes to solve conflicts.

We reran the unresolved conflict scenarios with these changes. The results are presented below:

TABLE III. UNRESOLVED CONFLICTS BEFORE PROMPT IMPROVEMENT

Model	Configuration	Collisions	LoS	Total
GPT-4o	Single Agent	4	4	8
	Single Agent + Exp	0	1	1
GPT-4o	Multi Agent	4	0	4
	Multi Agent + Exp	4	0	4
Llama3-70B	Multi Agent	2	3	5
	Multi Agent + Exp	2	3	5

TABLE IV. UNRESOLVED CONFLICTS AFTER PROMPT IMPROVEMENT

Model	Configuration	Collisions	LoS	Total
GPT-4o	Single Agent	0	0	0
	Single Agent with Exp	0	0	0
GPT-4o	Multi Agent	0	0	0
	Multi Agent with Exp	1	1	2
Llama3-70B	Multi Agent	4	0	4
	Multi Agent with Exp	0	0	0

The prompt improvements significantly enhanced performance across configurations. Notably, all configurations showed dramatic improvements except for the multi-agent

`Llama3-70B` without the experience library, which only improved from 5 to 4 unsuccessful scenarios. This minimal improvement underscores that hallucinations cannot be resolved through prompt engineering alone.

In the `GPT-4o` multi-agent with experience library, one loss of separation and one collision remained. In one case, the planner provided an instruction without numerical values, leading the executor to skip it. In another, a suboptimal plan caused a collision.

These remaining unresolved scenarios highlight the need for further refinement, especially in ensuring that instructions are explicit and correctly executed by agents.

D. Complexity of the traffic

Looking at Figure 8, it is evident that as the number of aircraft involved in a conflict increases, the success rate for resolving these conflicts declines. This outcome is expected as the more significant number of aircraft introduces more information that the large language model must process, which in turn impacts its performance.

Notably, the `GPT-4o` agent configurations maintain similar performance levels when dealing with conflicts involving two or three aircraft. There is a slight decrease in performance when the number of aircraft increases to four. For `Llama3:70B`, the performance is similar in multi-agent setups.

E. Computing resource constraints

Our testing was impacted by computing resource access, especially for model hosting and processing speed.

All models except `GPT-4o` were hosted by Groq, a cloud platform with a Language Processing Unit delivering around 1,000 tokens per second, but with strict token-per-minute and token-per-day limits. This constraint limited the number of parallel models and conflict scenarios we could test.

We also attempted to run `Llama3-70B` using Ollama on the TU Delft DelftBlue cluster [13], equipped with NVIDIA A100 GPUs. However, the inference speed on this platform was too slow for our needs.

Table V shows the substantial difference in token processing speeds between Groq and DelftBlue. Groq processes about 38,543 input tokens per second and 325 output tokens per second, whereas DelftBlue handles only 10.41 input tokens and 4.11 output tokens per second—3,700 times and 79 times slower, respectively. For complex scenarios like four-aircraft conflicts with prompts up to 8,000 tokens, DelftBlue would take over 10 minutes, limiting real-time responsiveness and scalability.

Groq's high token throughput allows fast processing even as prompt sizes grow, essential for efficient real-time operations and testing multiple scenarios concurrently.

TABLE V. COMPARISON OF TOKEN PROCESSING SPEEDS BETWEEN GROQ AND DELFTBLUE PLATFORMS

Platform	Input Tokens (tokens/s)	Output Tokens (tokens/s)
Groq	38543	325
DelftBlue	10	4

V. Conclusion

This study explored the application of large language models as embodied agents in air traffic control scenarios, focusing on their ability to resolve conflicts autonomously.

Our experiments with both open and closed-source models such as `Llama3:70B` and `GPT-4o` demonstrate the great potential of large language models embodied agents in performing air traffic control tasks. This new approach could reduce the gap between artificial and human situational awareness. We have demonstrated that it provides human-like reasoning with timely control instructions or recommendations.

The findings highlight that larger models outperform smaller models in complex conflict resolution scenarios. The incorporation of an experience library further aids in boosting efficiency by providing access to past conflict resolution insights, which is particularly beneficial for smaller models like `Llama3:70B`.

Moreover, the study has shown that multi-agent systems, where tasks are distributed among specialized agents, yield high success rates in resolving conflicts as well. This research paves the way for new research paths to apply language model-embodied agents in more complex tasks for air traffic management.

References

[1] G. Skaltsas, J. Rakas, and M. G. Karlaftis, "An analysis of air traffic controller-pilot miscommunication in the nextgen environment," *Journal of Air Transport Management*, vol. 27, pp. 46–51, 2013.

[2] U. Linz, S. Consulting, T. U. Braunschweig, U. P. D. Madrid, Z. H. F. A. Wissenschaften, Skyguide, and S. U. Z. F. P. Znanosti, "AI Situational Awareness Foundation for Advancing Automation." https://www.aisa-project.eu/, 2022. Accessed: 2024-09-01.

[3] Y. Zou and C. Borst, "Investigating transparency needs for supervising unmanned air traffic management systems," in *13th SESAR Innovation Days*, 2023.

[4] C. Hurt, A. Degas, A. Guibert, N. Durand, A. Ferreira, *et al.*, "Toward a more transparent and explainable conflict resolution algorithm for air traffic controllers," in *34th Conference of the European Association for Aviation Psychology*, European Association for Aviation Psychology, 2022.

[5] "Common crawl." https://commoncrawl.org/. Accessed: 2024-07-22.

[6] S. Abdulhak, W. Hubbard, K. Gopalakrishnan, and M. Z. Li, "Chatatc: Large language model-driven conversational agents for supporting strategic air traffic flow management," *arXiv preprint arXiv:2402.14850*, 2024.

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

[8] G. Jarry, P. Very, and R. Dalmau, "The effectiveness of large language models for textual analysis in air transportation," *EasyChair preprints*, 2024.

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

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

[11] Z. R. Tam, C.-K. Wu, Y.-L. Tsai, C.-Y. Lin, H. yi Lee, and Y.-N. Chen, "Let me speak freely? a study on the impact of format restrictions on performance of large language models," 2024.

[12] "Chroma." https://www.trychroma.com/. Accessed: 2024-07-22.

[13] Delft High Performance Computing Centre (DHPC), *DelftBlue Supercomputer (Phase 2)*, 2024. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.