Towards Siloed LLM-based Systems for Mission-critical Planning

Tyler D. Comisky[1], **Leslie N. Smith**[2], **Mark Roberts**[2], **Joshua Lovejoy**[3], **Avni Garg**[4], **Luke Nam**[3], **Adrian Li**[4], **Leora Samuels**[3]

[1]NAWCAD Lakehurst
[2]NCARAI, US Naval Research Laboratory, Washington, DC
[3]NREIP Intern at the US Naval Research Laboratory, Washington, DC
[4] SEAP Intern at the US Naval Research Laboratory, Washington, DC
tyler.d.comisky.civ@us.navy.mil, leslie.n.smith20.civ@us.navy.mil

Abstract

Adoption of Large Language Models (LLMs) is occurring at an accelerated pace. One area where LLMs currently fall short is in reliable and verifiable outputs. This makes it challenging to deploy LLMs to mission-critical environments, where reliability is required. Additional challenges occur when the environment must remain air-gapped, being entirely disconnected from the internet. We propose a framework for deploying siloed LLMs to these mission-critical environments using tools and verifiers. First, we demonstrate this framework's feasibility using commercial LLMs for travel planning. Then, we use this framework for the mission-critical application of generating aerial refueling schedules with air-gapped, open-source LLMs.

1 Introduction

Mission-critical applications represent some of the most demanding and consequential use cases for advanced software systems. These applications, ranging from healthcare decision support to financial trading systems and military operations, require flawless execution and absolute reliability. Failure is not an option when human lives, national security, or massive financial assets are at stake. However, the complexity of these domains, compounded by their often confidential nature, presents unique challenges for the development of software systems capable of meeting such stringent requirements.

Mission-critical systems, particularly those involving sensitive data, are frequently siloed to prevent data leakage. For instance, healthcare systems must protect patient information, financial institutions guard trading algorithms, and military operations shield their strategies from adversaries. This necessitates the use of open-source large language models (LLMs) implemented on secure, self-hosted hardware to meet both performance and confidentiality demands. Nevertheless, achieving 100% reliability remains a significant hurdle for such closed systems.

The emergence of LLMs and LLM-based agent systems offers new potential to enhance the effectiveness of mission-critical planning but deploying LLMs in mission-critical environments brings its own set of challenges. Despite their flexibility and scalability, LLMs can lack the precision, consistency, and accountability required in scenarios where every decision could have life-altering consequences.

This paper addresses these challenges by designing an LLM-based agent system specifically tailored for sensitive military air combat mission planning. We examine the strengths and limitations of existing LLM technology, pinpoint gaps in reliability, and propose methods to mitigate risks in this mission-critical environment. Our research aims to bring LLM-based systems closer to the robustness necessary for deployment in all high-stakes scenarios where reliability, privacy, and performance intersect.

Our paper is divided into two parts. In the first part, we investigate the TravelPlanner benchmark dataset (Xie et al. 2024), where we explore attaining high reliability without the requirement of keeping data siloed. By addressing these issues, we improved the reliability and performance of LLMs in complex planning tasks using ChatGPT where the data isn't kept private.

In the second part, we introduce a framework for planning assisted by siloed open-source LLMs for solving aerial refueling, which is a sub-component of air combat mission planning. For this task, we demonstrate our framework using Llama 3 70B. We demonstrate that by providing tools to the LLM that can algorithmically generate a plan, we mitigate the risk of hallucinations. First, the task of translating natural language to a structured format is less prone to hallucinations than generating a plan on its own. Second, if the LLM does hallucinate, since we expect a structured output, we can validate it more easily than an entire plan. Finally, most of the reasoning required to generate a plan is now outsourced to the tool, which is programmed to handle the complex tasks that LLMs are currently unable to reliably complete.

2 Related Works

Using LLMs to plan: Hao, et al. (Hao et al. 2024) present a novel LLM-based planning framework that formalizes complex multi-constraint planning problems as constrained satisfiability problems, achieving high success rates and strong zero-shot generalizability in real-world planning tasks. In our efforts to replicate this work, we found that a portion of the expertise needed to obtain high performance with the TravelPlanner dataset was hand coded into the system, reducing its generalizability.

"Can We Rely on LLM Agents to Draft Long-Horizon Plans? Let's Take TravelPlanner as an Example" Chen et al. (Chen et al. 2024) investigate the reliability of LLM agents in drafting long-horizon plans, using the TravelPlanner benchmark to evaluate their performance and explore potential improvements. They found that LLM agents struggle with lengthy and noisy contexts, often missing crucial information. Our work follows along similar lines but we are able to demonstrate greater improvements in performance.

A position paper (Kambhampati et al. 2024) argues that Large Language Models (LLMs) cannot perform planning or self-verification tasks independently but can significantly contribute to planning and reasoning tasks in LLM-Modulo frameworks as universal approximate knowledge sources. Key insights include the potential misconceptions about LLMs' capabilities in planning and reasoning and the proposal of a more nuanced role for LLMs in these tasks. Based on the content, novel future research directions could include: 1) Developing and evaluating specific LLM-Modulo frameworks for various planning and reasoning tasks, 2) Investigating the impact of different LLM architectures and training methods on their performance in LLM-Modulo frameworks, and 3) Exploring the integration of LLMs with other AI techniques, such as reinforcement learning or evolutionary algorithms, in planning and reasoning tasks.

The LLM-Module paper (Gundawar et al. 2024) provides a test of their proposed LLM-Modulo framework, where an LLM is paired with a complete set of sound verifiers that validate its output, re-prompting it if it fails. Valmeekam et al. (Valmeekam et al. 2024) evaluates and improves the planning and scheduling capabilities of OpenAI's Large Reasoning Models (LRMs), specifically o1-preview and o1-mini, demonstrating their superiority over autoregressive LLMs but highlighting high inference costs and lack of guarantees.

LLM Tool Use: There has been substantial work on developing frameworks and interfaces for integrating tools with LLMs, as described in some recent survey papers (Qu et al. 2024; Wang et al. 2024b). Tools are a way for LLMs to call external functions that provide additional capabilities to LLMs in performing a wider variety of tasks (Shen 2024). These tools can be Python functions whose purpose is passed to the LLM via prompt, and can then be called by the LLM with the appropriate parameters passed to it. The ReAct framework (Yao et al. 2023) proposes an interface that enables reasoning and acting in LLMs, enabling seamless integration of multiple tools. Their framework demonstrates improved performance and flexibility across various tasks.

Deploying LLMs: There has been some research into deploying LLMs in sectors such as healthcare (Thirunavukarasu et al. 2023; Yang et al. 2023), where the accuracy of the LLM is critical to its use. Outside of healthcare, there are limited studies of deploying LLMs in mission critical environments (Esposito et al. 2024), where its failure would result in severe consequences. One of the traits of a mission critical system is its ability to be vigorously tested to ensure its reliability under all conditions. LLMs are inherently unreliable, due to their nondeterministic structure. As such, LLMs must be used in a way such that their outputs can be easily and automatically verifiable. Our proposed framework enables this verification by forcing the LLM to output to a Pydantic model, which is a format whose structure can be enforced.

Automated Fuel Planning: Previous work in aerial refueling planning has been conducted for various applications for military and civilian use (Huang et al. 2024; Panos 2007). Organizations such as MITRE and Kessel Run have built tools such as JIGSAW for the US Air Force to aid in building refueling schedules (Altner et al. 2024). An earlier iteration of MITRE's JIGSAW implementation used linear programming techniques, but they have recently moved over to neighborhood search methods, as we describe. However, similar to other solutions, their algorithm assigns tankers to pre-defined refueling requests. We allow the refueling requests to be dynamically generated to optimize for fuel consumption. Our method provides the unique ability to use natural language as input, which makes using our tool much easier than its predecessors.

3 TravelPlanner

TravelPlanner (Xie et al. 2024) is a benchmark for creating travel itineraries provided a start city, destination city/cities, number of days, number of travelers, and various other constraints (e.g., cuisines, pets, minimum nights). However, even the most sophisticated LLMs at the time of the paper's publication had trouble getting even modest performance on the benchmark, as the highest pass rate was gpt-4 at **0.6%**. Furthermore, mission planning is substantially more complex than the travel itineraries in TravelPlanner so we consider this only a first step towards achieving reliable mission planning.

We aimed to increase the accuracy of TravelPlanner using a combination of the two following approaches:

1. Prompt engineering methods for better parsing arguments and guiding future LLM actions
2. Callable helper functions that reason for the LLM and output observations

In our experiments we found that the LLM-based system demonstrated several specific failings when attempting to make travel plans. In this section we describe the limitations and the solutions we found to fix the specific problems we encountered.

Running Over Budget

Problem: LLMs have trouble with numerical reasoning, especially when it comes to large numbers or keeping track of the current day of a trip that spans multiple days (Ahn et al. 2024). In one test case, we asked for a 3-day itinerary from St. Petersburg to Appleton within a \$1,200 budget, the LLM generated a plan that ran over budget and was too short. Other 3-day itinerary requests, with different cities and new budgets, produced similarly expensive and short itineraries. Often, the LLM would state that a plan costs X

dollars, and when calculating manually would reveal that it costs Y dollars, where $X < Y$.

Solution: We developed a hybrid approach combining prompt engineering and helper functions which provided the most effective solution to address the LLM's inability to accurately calculate costs. Initially, the **CostEnquiry[SUBPLAN]** function processes a JavaScript Object Notation (JSON) subplan to calculate the itinerary cost. This cost is then sent to the LLM, which is supposed to compare it with the budget specified in the input prompt. However, at this stage, the LLM consistently failed to recognize when it exceeded the budget.

To resolve this issue, we implemented a crucial modification: instead of relying on the LLM to compare numbers, we enhanced the CostEnquiry function to perform the comparison between the output cost and budget. Following this comparison, CostEnquiry sends an observation message (as part of the ReAct framework) to inform the LLM whether it has stayed within budget. This modification dramatically improved the LLM's performance, virtually eliminating miscalculations in travel plans.

In edge cases where no travel itinerary exists within the given budget constraints, the LLM occasionally stated that an over-budget plan fits within the budget, then proceeding to call the **Finish** function to send its final plan to the user. However, in the majority of cases, the enhanced **CostEnquiry** function proved highly effective. Furthermore, by carefully engineering the observation message's prompt, we encouraged the LLM to explore more cost-effective alternatives in subsequent iterations. Overall, this hybrid approach, combining prompt engineering with algorithmic helper functions, successfully addressed this challenge.

Following these improvements, we shifted our focus to simplified 1-day plans for the majority of our tests, departing from the easy (3-day), medium (5-day), and hard (7-day) tests outlined in the original TravelPlanner paper (Xie et al. 2024). This decision was motivated by the observation that LLMs plan travel itineraries iteratively on a day-by-day basis. For instance, when creating a 5-day plan, the LLM assigns events to Day 1, then proceeds to Day 2 only after confirming that the first day's events are valid and within the given constraints. However, due to their limitations in formulating long-term, multi-step plans, LLMs often lose track of the current day they are planning, leading to the premature output of faulty plans.

By focusing on 1-day plans, we could more effectively evaluate the performance of our helper functions, as this approach eliminates the possibility of the LLM forgetting the current day and consequently producing an erroneous plan. It is worth noting that decomposing long-term plans into smaller subplans is a well-documented approach, featured in other LLM-based planners such as DELTA (Liu et al. 2024) and RobLM (Chalvatzaki et al. 2023), which have demonstrated significant improvements. For future work on longer travel itineraries, further exploration of these task decomposition frameworks could prove valuable in preventing temporal hallucinations. One such example is TwoStep, which uses two LLM agents to decompose tasks by approximating human intuition. (Singh, Traum, and Thomason 2024). The remainder of this section will concentrate on the benefits of prompt engineering and helper functions, rather than looking into the implementation of these broader task decomposition frameworks.

The Cheapest Option Cannot Be Found

Problem: Sticking with the previous example of a 1-day, 1-person itinerary from St. Petersburg to Appleton, it is possible to create an itinerary under \$656, which is almost the cheapest plan that either a human or an LLM planner can make. This test case was meant to evaluate the LLM's ability to find the cheapest amenity (e.g., accommodation, flight, restaurant) that meets all the input constraints.

Unfortunately, the LLM instead decided to increase the subplan's cost, suggesting that the LLM cannot identify the cheapest accommodation from the input prompt.

Solution: One prompt engineering method that showed mixed results was keeping track of the cheapest plan that the LLM had generated so far. For certain trials, when the LLM could not generate a plan that fit the budget constraint, it would output the best plan it was able to make.

This was a significant change from previous trials. When the LLM failed to find a plan that met the budget constraint, it would output a random plan generated without any consideration of its previous thoughts. These plans would also often be much pricier than the budget. For this reason, this approach of keeping the "cheapest plan so far" would still run over budget, but not at a ludicrously large scale.

The best approach for finding the cheapest option from a list, however, is to prompt the LLM to call a helper function to find the cheapest amenity in a city within the provided constraints.

Premature Option Exhaustion

Problem Similar to the previous issue of LLMs being unable to find the cheapest option that satisfies all constraints from the input, LLMs would also incorrectly declare that they have searched and "exhausted all possible options" from the input JSON data. The LLM would proceed to output a finalized plan that fails to meet constraints and stated that no plan could possibly satisfy them. However, upon planning by hand, we know that it is possible to create a travel plan that meets all the given constraints.

Solution: ExpeL (Zhao et al. 2023) is a framework built on ReAct (Yao et al. 2023) that gathers "experiences" from test cases and extracts "insights" from them. Consequently, when running new test cases that the ExpelAgent has never seen before, it draws insights from its previous experiences to solve the presented question.

The HotpotQA benchmark (Yang et al. 2018) evaluates an LLM agent's ability to answer questions given a selection of Wikipedia articles. The ReAct Agent incorrectly found no answers from its selection of Wikipedia articles when one does exist, similar to the TravelPlanner test cases where an optimal accommodation exists. On the other hand, the ExpelAgent learned from past experiences that the answer may already be "in the observations already made", and manages to find the answer afterwards.

An Algorithmic Approach

Problem: As covered in the previous sections, using the LLM to manage planning is highly inaccurate, even with the help of tools. LLMs hallucinate frequently, exhaust options prematurely, and ignore instructions. Attempts to plan travel itineraries using an LLM as a planner are extremely inaccurate, which is an observation consistent with other research papers (Stechly, Valmeekam, and Kambhampati 2024).

Solution: Owing to its poor performance, we repurposed the LLM as a natural language interface, and we instead implemented a hard-coded algorithm for creating a plan that follows all the constraints. This two-step approach of combining an LLM with a traditional algorithm has been credited with improving plan outputs for LLM-based planners. For some agents, LLMs convert natural language to PDDL domains and actions (Guan et al. 2023; Oswald et al. 2024; Silver et al. 2023). Beyond PDDL, LLMs can convert natural language to Python (Wang et al. 2024a), JavaScript (Schäfer et al. 2023), mathematical notation (Li et al. 2024), and other custom languages (Ji et al. 2024).

For our algorithmic approach, the LLM is tasked with generating an input configuration in the form of a JSON file. The planning algorithm proceeds to use this JSON file as its input. To plan trips, the algorithm must first select which cities to visit in the case of multi-city trips. It will then look at every city within a state, and determine the cost to visit that city taking into consideration the constraints given. Some cities can be impossible to reach by other cities on certain days, so the algorithm would choose the cheapest cities that can satisfy a round trip.

To choose accommodations, restaurants, attractions, and transportation options, the algorithm uses a simple brute-force search. For accommodations, restaurants, and attractions, the algorithm searches through all accommodations/restaurants/attractions in the city and finds the cheapest one that fits all constraints. For planning transportation, the algorithm chooses the cheaper option between self-driving and a combination of flights and taxis.

The algorithmic approach increased accuracy to 90%, a major improvement from the original paper's 0.6% accuracy (Xie et al. 2024). The LLM still has trouble generating the input JSON file from natural language due to errors in the benchmark construction, either outputting a blank response or formatting the input incorrectly, both of which would cause the algorithm to be unable to create any travel plan. The algorithm itself also is not perfect, as it can have trouble finding enough valid cities or keeping its plan within the provided budget. Despite these flaws, the high success rate serves as a baseline as to what an LLM-based planner should be able to achieve. In the long term, LLM-assisted planning applications should implement a similar approach to what is described here: using an LLM to parse natural language into an input configuration for a traditional algorithm.

TravelPlanner Summary

In the process of using LLMs to devise travel itineraries, we encountered several challenges, particularly in relation to numerical reasoning and keeping track of the current day for multi-day trips. The LLM often generated plans that exceeded the budget or were too short, and it struggled to accurately calculate costs. In addition, we encountered several challenges in relation to the LLM incorrectly declaring that it has exhausted all possible options from the input JSON data and failing to meet constraints. Additionally, we found that using LLMs to manage planning is highly inaccurate due to their frequent hallucinations, premature option exhaustion, and disregard for instructions.

To address these issues, we pushed numerical tasks to the tools and implemented an algorithmic approach that primarily used the LLM as a natural language interface. The LLM generates an input configuration in the form of a JSON file, which is then used as input for the planning algorithm. The algorithm selects cities to visit, determines the cost to visit each city, and uses a search to choose accommodations, restaurants, attractions, and transportation options. This algorithmic approach significantly improved the accuracy of the travel itineraries, increasing it to 90% from the original paper's 0.6% accuracy.

The lessons learned from these experiments and experiences include the importance of using a hybrid approach combining prompt engineering and algorithmic helper functions to address the LLM's limitations in numerical reasoning and keeping track of the current day for multi-day trips. Additionally, decomposing long-term plans into smaller subplans proved to be a valuable strategy for improving the performance of LLM-based planners. This work lays the foundation for deploying LLMs into mission-critical environments, where accuracy is essential for its adoption.

4 Aerial Refuel Planning

Our next step is to apply the lessons learned from TravelPlanner to the automated generation of aerial refueling schedules, which is one potential application of using LLMs as a planner in mission-critical applications. Aerial refueling scheduling is the process of allotting groups of receivers (i.e., aircraft) to tankers to ensure all receivers land with a certain amount of fuel while satisfying additional time and weight constraints. These schedules have traditionally been drafted by hand, a process which could take hours (Altner et al. 2024).

Our goal in this work was to achieve a highly reliable fuel planning application that can run confidentially with open source LLMs on our own air-gapped servers. This application aims to shorten the mission planning time and optimize fuel consumption for jets and tankers in air combat missions. It leverages LLMs to process natural language scenarios, generating an optimized refueling plan with enhanced efficiency and accuracy.

Architecture

The architecture of our system is made up of the input, output, and agent framework. Figure 1 displays the pipeline of

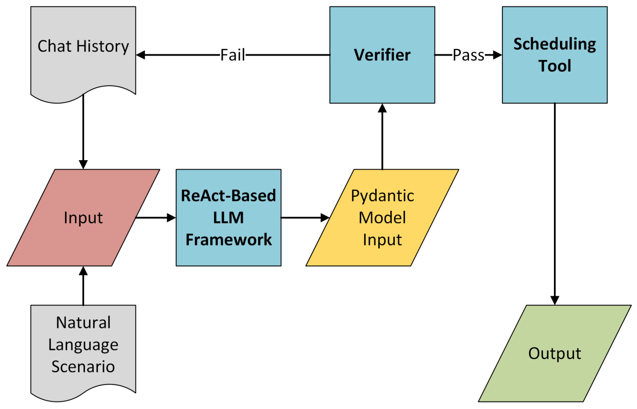

Figure 1: Pipeline for the aerial refueling planning tool.

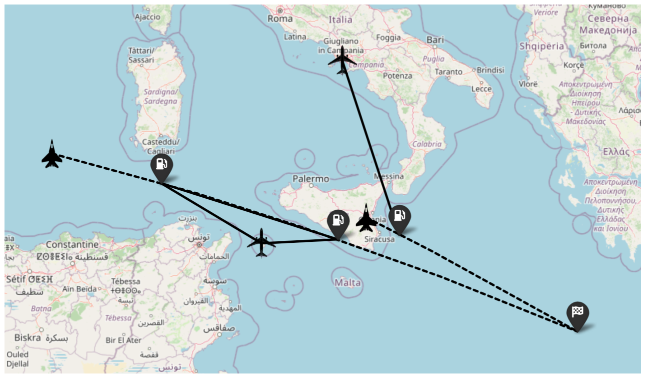

Figure 2: Example aerial refueling plan ($receivers = 2, tankers = 2$) generated by neighborhood search.

the components, which is described in the following sections.

Input: The input component provides not only input from the user but also additional information such as the system prompt and chat history. First, the input from the human is a natural language description of the information required to generate a refueling schedule. Examples of this can be found in Appendix A. The system prompt contains a description of the LLM task and describes how the LLM should respond using the ReAct method (Yao et al. 2023). This prompt can also be found in Appendix A. Finally, the chat history is added to provide the LLM with previous steps it has already taken, and any additional information from the user.

Agent-based planning framework: We developed an LLM agent to serve as an interface between the user and scheduling algorithm. Its purpose is to translate a natural language scenario description to a structured input configuration, that will then be passed to the scheduling algorithm as a set of parameters. This LLM agent is able to use any locally hosted model. We used Llama 3 70B to test this framework. To develop this agent, we used the LlamaIndex framework because it provides several components, such as chat history, output parsers, and a ReAct agent (Liu 2022).

Pydantic is a data validation library for Python. It allows users to define the structure of an object using a BaseModel. The LLM populates all the fields it can from the given scenario. The output parser is able to determine if fields are missing, and will prompt the user for more information if it finds out that is the case. However, the output parser is not able to determine whether populated fields are semantically correct. So, after the output parser, we call a manually written validation function that checks if all locations and receivers/tankers are actually valid. In theory, if the generated configuration is not valid on the first try, by giving the LLM specific messages as to why the configuration is not valid so it would be able to correct itself. If the LLM is still unable to correct the error, the human is then given the option to manually fill in the erroneous fields.

Prior methods have shown that LLMs are better at solving reasoning tasks when they separate the task in several smaller subtasks, with one such method being ReAct. (Yao et al. 2023) Using this method, we prompt the LLM to identify the aircrafts, origins, destinations, and several fuel-related constraints that the plan must adhere to. Instead of trying to take all of this information and translate into a structured format in a single step, the LLM iteratively identifies each item individually, and adds it to the output. This provides a result that is much less prone to hallucinations.

Output: The final result of the LLM's output is a Python script which was generated using a Pydantic model. This model contains classes for every object that is required for each plan, including the Trip, Receiving Aircraft, Tanker Aircraft, and Constraints. Each class contains fields specific to that object. This ensures that when the LLM is creating these classes, it is being provided with the correct fields. If the LLM fails to create a class, then the error output by the Python program is passed back into the LLM, and it tries again. By structuring the output of the LLM in a Pydantic model, we are able to validate the LLM's output at each step of the process, and conduct a final verification once the entire Pydantic model is completed to ensure all of the necessary information for a plan is included. This is a necessary step to enable the deployment of LLMs to mission-critical environments, because it allows for an automatic verficiation of the LLMs output. As a last resort, there is an option for a human to manually review the Pydantic model before it is passed to the planning algorithm.

Planning Algorithm (The Tool)

We developed this agentic planning framework to assist in the creation of aerial refueling schedules. This problem is a variation on the vehicle routing problem (Funke, Grünert, and Irnich 2005). Neighborhood search is the optimization technique typically used to solve such problems. It relies on starting with some initial solution which may or may not be feasible, and iteratively transforming it to minimize cost. The cost is determined with an objective function which includes penalties for infeasible solutions.

In our approach, we introduce the concept of a Receiver-Group (RG), which consists of one type of receiver, the quantity of that receiver type, the path the receiver follows including an ID, start coordinate, and end coordinate, and the time the group will take off from the starting base. Additionally, there is the option to change the distance between the waypoints that will be generated along the specified path, where waypoints are the refueling points at which the tanker intercepts the receiver groups. The waypoints generated by the ReceiverGroup are created as a LinkedList to easily access neighboring waypoints and are stored in a dictionary, indexed in order by time. Each receiver type contains specifications and the calculated starting and ending fuel to perform required missions.

Once the algorithm is complete, it produces a CSV file containing the time and distance of each refueling point along with the corresponding tanker and receiver aircraft. Then we take this data and display it on a map for easy visualization (see Figure 2). This acts as a final sanity check to ensure the plan makes sense before it is implemented.

The initial algorithm we developed scheduled receiver refuelings one at a time, starting from the end fuel level and ending at the start fuel level. This brute force method had several limitations. The main limitation is that we assume the same tanker escorts a RG from origin to destination, which is most likely not the case in real-world scenarios. In order to fix this, we developed an algorithm in which tankers can start at a distinct location from other tankers and RGs, meet a RG at a rendezvous point along the RG path, perform the refueling, and either return to its base or perform another refueling, depending on which saves more fuel. This algorithm employs a neighborhood search described below.

Another limitation of this backwards iterative approach is that there are various edge cases to consider when attempting to find the intersection between the forward burning line and the backward refueling line. Sometimes, the window might be too small and not able to fit all receivers in the RG. In order to fix this, we would need to decrease an offload somewhere else. In the end, we were not able to achieve 100% correctness, and decided to focus our efforts on developing the much more robust neighborhood search algorithm.

Fuel Planning with Neighborhood Search

Our previous approach for fuel planning was limited because it required the tanker to escort the receiver groups from start to end. Many times, tankers are located at different locations than receiver groups and will fly to meet them at scheduled refueling points. Our goal was to develop a more robust algorithm that planned these various refueling stations, providing greater flexibility when making these refueling schedules.

To integrate scheduling receivers and tankers from multiple locations, multiple changes had to be made to the program. These multi-input configurations consist of locations, tankers, and receiver groups: groups of the same type of receiver traveling from one location to another. New tools had to be made so the ReAct agent could initialize these locations, receiver groups, and tankers. Additionally, an input configuration class had to be made to represent these missions. This input configuration would be used to create a scenario Pydantic model Python file, which contained all the information necessary to run the program.

ReAct Agent Integration: To address the challenge of accurately extracting data from natural language inputs, a ReAct Agent class was created utilizing the ReAct framework (Yao et al. 2023) and a set of tools to iteratively build the configuration. This new class demonstrated superior performance in processing natural language input and accurately filling out data for each constraint compared to the original Agent class.

Compared to the original Agent, the ReAct Agent displayed improved performance in extracting correct numbers even when the scenario was worded differently. This effectively solved the issue of constraint inaccuracies and improved the mission planning process within the fuel planning project. The ReAct Agent's success in correctly interpreting data significantly improved the efficiency and accuracy of aerial refueling schedules, contributing to enhanced mission outcomes and resource management.

Fuel Planning Summary

We developed a framework that uses state-of-the-art open-source LLMs to extract relevant information from natural language and generate a structured configuration object, using a Pydantic-defined JSON schema and a ReAct Agent. Additionally, we designed a neighborhood search algorithm that will not only create a feasible plan but also an optimal plan with the goal of minimizing the amount of fuel used. The proposed framework brings us one step closer to safely deploying LLMs in mission-critical environments.

5 Conclusions

We explored the use of Generative AI and LLMs in creating travel itineraries and aerial refueling schedules, highlighting both their potential and limitations. Through a series of experiments and case studies, we have demonstrated that LLMs can be used as a natural language interface, with a traditional algorithm employed for planning purposes. This hybrid approach has shown significant improvements in the accuracy of travel itineraries, increasing it from the original 0.6% accuracy to 90%. However, LLMs still face challenges in generating input JSON files from natural language and in creating perfect plans due to difficulties in finding valid cities or keeping within budget constraints.

In the context of a mission-critical application such as aerial refueling planning, we have developed an LLM agent framework using LlamaIndex, which enables the translation of natural language descriptions into structured input configurations. This framework leverages Pydantic models and a ReAct-based agent to minimize hallucinations, reduce their impact, and improve the accuracy of the generated configurations. The output of the LLM is a Python script containing classes for each object required for the plan, ensuring that the LLM populates the correct fields. Further validation and verification were conducted to ensure the accuracy and completeness of the generated configuration. All of this was performed on a siloed server, ensuring no data leakage.

In summary, we first demonstrated a dramatic increase in accuracy is possible with a commercial LLM by employing careful prompt engineering and a hybrid system that includes software tools and verification functions. While we did not achieve the 100% accuracy required of mission-critical applications, our work demonstrated a path forward. Second, we demonstrated a siloed mission-critical application of aerial fuel planning with open-source LLMs on our air-gapped servers. Our progress illustrates a means for creating LLM-based hybrid systems for any mission-critical application where data security is paramount.

References

Ahn, J.; Verma, R.; Lou, R.; Liu, D.; Zhang, R.; and Yin, W. 2024. Large Language Models for Mathematical Reasoning: Progresses and Challenges. arXiv:2402.00157.

Altner, D. S.; Armstrong, I. A.; Pusateri, A.; Armstrong, A. M.; and Bennett, R. P. 2024. US Air Force Aerial Refueling Optimization. *Military Operations Research*, 29(2): 19–36.

Chalvatzaki, G.; Younes, A.; Nandha, D.; Le, A. T.; Ribeiro, L. F.; and Gurevych, I. 2023. Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning. *Frontiers in Robotics and AI*, 10: 1221739.

Chen, Y.; Pesaranghader, A.; Sadhu, T.; and Yi, D. H. 2024. Can We Rely on LLM Agents to Draft Long-Horizon Plans? Let's Take TravelPlanner as an Example. *arXiv preprint arXiv:2408.06318*.

Esposito, M.; Palagiano, F.; Lenarduzzi, V.; and Taibi, D. 2024. Beyond Words: On Large Language Models Actionability in Mission-Critical Risk Analysis. arXiv:2406.10273.

Funke, B.; Grünert, T.; and Irnich, S. 2005. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of heuristics*, 11: 267–306.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning.

Gundawar, A.; Verma, M.; Stechly, K.; Valmeekam, K.; Guan, L.; Bhambri, S.; and Kambhampati, S. 2024. Robust Scheduling with the LLM-Modulo Framework: A Case Study in TravelPlanner. *submited to NeurIPS 2024*.

Hao, Y.; Chen, Y.; Zhang, Y.; and Fan, C. 2024. Large Language Models Can Plan Your Travels Rigorously with Formal Verification Tools. *arXiv preprint arXiv:2404.11891*.

Huang, J.; Xing, J.; Ren, J.; Quan, Q.; and Zhang, Y. 2024. Aerial refueling scheduling of multi-receiver and multi-tanker under spatial-temporal constraints for forest firefighting. *Chinese Journal of Aeronautics*, 37(5): 71–91.

Ji, Z.; Wu, D.; Ma, P.; Li, Z.; and Wang, S. 2024. Testing and Understanding Erroneous Planning in LLM Agents through Synthesized User Inputs.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Stechly, K.; Verma, M.; Bhambri, S.; Saldyt, L.; and Murthy, A. 2024. LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. *arXiv preprint arXiv:2402.01817*.

Kim, J.; Song, B. D.; and Morrison, J. R. 2013. On the scheduling of systems of UAVs and fuel service stations for long-term mission fulfillment. *Journal of Intelligent & Robotic Systems*, 70: 347–359.

Li, Z.; Hua, W.; Wang, H.; Zhu, H.; and Zhang, Y. 2024. Formal-LLM: Integrating Formal Language and Natural Language for Controllable LLM-based Agents.

Liu, J. 2022. LlamaIndex.

Liu, Y.; Palmieri, L.; Koch, S.; Georgievski, I.; and Aiello, M. 2024. DELTA: Decomposed Efficient Long-Term Robot Task Planning using Large Language Models.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large language models as planning domain generators. In *34th International Conference on Automated Planning and Scheduling*. openreview.net.

Panos, D. C. 2007. *Approximate dynamic programming and aerial refueling*. Ph.D. thesis, Citeseer.

Qu, C.; Dai, S.; Wei, X.; Cai, H.; Wang, S.; Yin, D.; Xu, J.; and Wen, J.-R. 2024. Tool Learning with Large Language Models: A Survey. *arXiv preprint arXiv:2405.17935*.

Schäfer, M.; Nadi, S.; Eghbali, A.; and Tip, F. 2023. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation.

Shen, Z. 2024. LLM With Tools: A Survey. arXiv:2409.18807.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J. B.; Kaelbling, L. P.; and Katz, M. 2023. Generalized Planning in PDDL Domains with Pretrained Large Language Models.

Singh, I.; Traum, D.; and Thomason, J. 2024. TwoStep: Multi-agent Task Planning using Classical Planners and Large Language Models. arXiv:2403.17246.

Stechly, K.; Valmeekam, K.; and Kambhampati, S. 2024. On the Self-Verification Limitations of Large Language Models on Reasoning and Planning Tasks.

Thirunavukarasu, A. J.; Ting, D. S. J.; Elangovan, K.; Gutierrez, L.; Tan, T. F.; and Ting, D. S. W. 2023. Large language models in medicine. *Nature medicine*, 29(8): 1930–1940.

Valmeekam, K.; Stechly, K.; Gundawar, A.; and Kambhampati, S. 2024. Planning in Strawberry Fields: Evaluating and Improving the Planning and Scheduling Capabilities of LRM o1. *arXiv preprint arXiv:2410.02162*.

Wang, X.; Chen, Y.; Yuan, L.; Zhang, Y.; Li, Y.; Peng, H.; and Ji, H. 2024a. Executable Code Actions Elicit Better LLM Agents.

Wang, Z.; Cheng, Z.; Zhu, H.; Fried, D.; and Neubig, G. 2024b. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*.

Xie, J.; Zhang, K.; Chen, J.; Zhu, T.; Lou, R.; Tian, Y.; Xiao, Y.; and Su, Y. 2024. TravelPlanner: A Benchmark for Real-World Planning with Language Agents.

Yang, R.; Tan, T. F.; Lu, W.; Thirunavukarasu, A. J.; Ting, D. S. W.; and Liu, N. 2023. Large language models in health care: Development, applications, and challenges. *Health Care Science*, 2(4): 255–263.

Yang, Z.; Qi, P.; Zhang, S.; Bengio, Y.; Cohen, W. W.; Salakhutdinov, R.; and Manning, C. D. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models.

Zhao, A.; Huang, D.; Xu, Q.; Lin, M.; Liu, Y.-J.; and Huang, G. 2023. ExpeL: LLM Agents Are Experiential Learners.

A Fuel Planner LLM Prompts

System Prompts

Here is the system prompt being used for the ReAct-based LLM agent:

You are a large language model designed to answer questions for aerial refuel scheduling based on a scenario (provided in subsequent prompts) and chat history.

Tools You have access to a wide variety of tools. You are responsible for using the tools in any sequence you deem appropriate to complete the task at hand. This may require breaking the task into subtasks and using different tools to complete each subtask.

You have access to the following tools: tool_desc

Output Format To answer the question, please use the following format.

"' Thought: I need to use a tool to help me answer the question. Action: tool name (one of tool_names) if using a tool. Action Input: the input to the tool, in a JSON format representing the kwargs (e.g. "input": "hello world", "num_beams": 5) "'

Please ALWAYS start with a Thought.

Please use a valid JSON format for the Action Input. Do NOT do this 'input': 'hello world', 'num_beams': 5.

If this format is used, the user will respond in the following format:

"' Observation: tool response "'

You should keep repeating the above format until you have enough information to answer the question without using any more tools. At that point, you MUST respond in the one of the following two formats:

"' Thought: I can answer without using any more tools. Answer: [your answer here] "'

"' Thought: I cannot answer the question with the provided tools. Answer: Sorry, I cannot answer your query. "'

Additional Rules - The answer MUST contain a sequence of bullet points that explain how you arrived at the answer. This can include aspects of the previous conversation history. - You MUST obey the function signature of each tool. Do NOT pass in no arguments if the function expects arguments. - You do not need to use the tools if they are not needed to answer the question. Only use the tools provided, and no other. - If the answer is not provided in the given scenerio, prompt the user for more information.

Additional information - Rendezvous occurs at time t = 0. Therefore, if something must occur within the first x minutes of rendezvous, the time range will be [0, x]. - If something must occur within x nautical miles of the destination and the total distance is d, then the distance range will be start_distance = 0 and end_distance = d - x. - All units of time must be in minutes. For example, if a jet burns fuel at b lbs/hour, you must list this as (b / 60) lbs/min.

Current Conversation Below is the current conversation consisting of interleaving human and assistant messages.

Example Scenarios

Single Origin and Destination Scenario: Three F-35C fighter jets (Grizzly 11, 12, and 13) are flying from Oak Harbor, WA, to Honolulu. Each jet starts with 18,000 lbs of fuel and has a maximum fuel capacity of 22,000 lbs. The total distance is 2,700 nm. Jets must maintain a minimum of 13,000 lbs of fuel until within 450 nm of Honolulu. Upon landing, each jet needs at least 5,000 lbs of fuel. Jets burn fuel at 6,800 lbs/hour while cruising at 410 knots. A KC-135R tanker (Texaco 21) carries 80,000 lbs of fuel and can refuel one jet at a time. Refueling takes 15 minutes for 11,000 lbs of fuel. During refueling, jets burn fuel at 1,900 lbs/15 minutes. There's a 3-minute transition between refueling different jets. Each jet must test refueling by taking on at least 1,000 lbs of fuel from Texaco 21 within the first 30 minutes after rendezvous.

Multiple Origins, Single Destination Scenario: The locations for the mission are sigonella at 37.405 N 14.922 E, cvn75 at 38.822 N 6.044 E, naples at 40.836 N 14.249 E, and pantelleria at 36.831 N 11.945 E. The mission ends at 34.812 N 20.867 E. There are tanker bases at naples and pantelleria. There is a receiver group of 1 f-22a going from sigonella to the end, and another receiver group of 2 f-35c receivers going from cvn75 to the end. There is a kc-135r tanker at naples and a kc-46 tanker at pantelleria.