An Agentic Orchestration System for Heliophysics Tasks

Russell Spiewak
Trillium Technologies Inc
Frisco, TX 75034
`russell.spiewak@trillium.tech`

Kevin Lee
Frontier Development Lab
Frisco, TX 75034
`kevinlee69720@g.ucla.edu`

James Walsh
Department of Engineering
University of Cambridge
Cambridge, UK
`jw2250@cam.ac.uk`

Abstract

We propose an agentic orchestration system for heliophysics tasks. Heliophysics research faces significant challenges in synthesizing vast, heterogeneous datasets from multiple ground-based observatories and space missions, with traditional methodologies remaining largely manual and siloed. This paper presents an agentic orchestration system that addresses these limitations by enabling integration and interaction between computational models across heliophysics research. The system employs Large Language Model-based agents structured according to established design patterns. Our implementation leverages state-of-the-art orchestration primitives, specifically Anthropic's Model Context Protocol for tool description and Google's Agent Development Kit for agent-to-agent communication. The system incorporates domain-specific tools ranging from ionospheric models to solar surface simulations, augmented by Retrieval Augmented Generation containing heliophysics literature and worked examples. Valuation was conducted through demonstrated capabilities in ionospheric modeling, solar surface analysis, automated pipeline generation, and tool discovery. Our system autonomously generates data pipelines, creating and managing computational infrastructure, all whilst requiring human oversight for critical decisions. The system reduces prototyping time from months to minutes, providing natural language access to sophisticated heliophysics simulations and machine learning models. This work establishes a first-attempt for accelerated scientific discovery in heliophysics by improving access to computational tools and enabling rapid hypothesis testing through automated workflow orchestration.

1 Introduction

The field of heliophysics is characterized by vast, multi-instrument, and multiscale datasets. This wealth of information originates from a global network of ground-based observatories and an ever-growing fleet of space missions, each providing additional perspectives and a plethora of new data. The data spans an immense range of spatial and temporal scales, capturing everything from short-lived, high-energy events like solar flares that can occur over seconds, to long-term phenomena like the 11-year solar cycle, and covers imaging active regions on the surface of the Sun, measuring the electron content in the ionosphere, and beyond. Despite this abundance of information, traditional research methodologies are often manual and siloed, and great effort is typically required to synthesize these complex, heterogeneous information streams (notable examples include significant efforts to transform Solar Dynamics Observatory (SDO) mission data into machine learning-ready datasets [1, 2]). Researchers typically work with small, disconnected subsets of data from one or two instruments at a time, making it difficult to construct a comprehensive understanding of highly interconnected phenomena. This fragmented approach leads to inefficiencies, missed connections between disparate observations, and a bottleneck in generating new insights. It would be helpful to provide the heliophysics data processing community with the option of quick natural language conversation with an experiment represented by a Jupyter notebook.

Here, we introduce an agentic orchestration system for heliophysics, designed to address these challenges by establishing a new approach to seamless integration, interaction, and fine-tuning of models developed across multiple teams in various research organizations. This approach employs AI agents to interact with diverse models, and includes adapting variable input, running complex workflows, and assisting in the production of near state-of-the-art level outputs. This system creates an adaptable environment where AI agents can, with human operator oversight, autonomously manage complex workflows, assisting common new occurrences in the field, such as fine-tuning models for specific objectives. The implementation of this system involves adopting and adapting recent orchestration primitives such as Anthropic's Model Context Protocol (MCP) [3] tool-description and Agent-to-Agent communication through Google's Agent Development Kit (ADK) [4]. The agentic orchestration system for heliophysics enhances flexibility, interoperability, and aids scientific discovery. Although these orchestration primitives provide the technical foundation for our system, understanding the current landscape of machine learning applications in heliophysics is essential to appreciate the transformative potential of agentic orchestration in this domain.

2 Background and Related Work

Machine learning has emerged as a powerful analysis solution, offering a new approach to tackling long-standing challenges in the field of heliophysics. Convolutional Neural Networks have proven particularly effective for analyzing solar imagery. The translationally invariant filters in convolutional layers that give them the ability to automatically learn features from visual data make them ideal for tasks such as detecting and classifying sunspots [5–8]. For time-series analysis and forecasting, such as predicting the intensity and timing of solar flares, Recurrent Neural Networks and variants such as Long Short-Term Memory networks are frequently employed. Due to their recurrence and "memory" states, these models excel at identifying patterns and dependencies within sequential data, critical for evaluating evolution of solar events [9, 10]. Finally, transformer models have also been recently applied in heliophysics space [11, 12]. Due to their self-attention mechanisms, transformers can weigh the importance of detecting different features corresponding to, say, solar active regions.

In the natural language modality, Large Language Models (LLMs) have recently evolved to consume a series of input prompts that are broken down into a series of tokens. The LLM then uses its learned patterns to predict the most probable next token, thus building a coherent response one word at a time. An agent then is a unit that uses specialized, structured prompts and tools to guide the LLM's reasoning and action, allowing it to interact with external data sources or perform complex tasks beyond simple text generation. Agents can interact with each other by taking the output of one agent as the input for another, creating a workflow where they collaborate to solve complex, multi-step problems that would be impossible for a single LLM to handle alone. This allows for a robust and dynamic system where each agent specializes in a specific task, such as data analysis, forecasting, or classification, and collectively works towards a unified objective. For example, an agentic system may be tasked with running simulations of physical phenomena, typically incurring a programming environment restriction. Such a request could include a data search agent to locate appropriate data, an infrastructure agent to query available compute resources (e.g., cloud, high performance computers, etc.), a scripting agent to write simulation scripts in the desired programming language and to build an appropriate execution environment, and an analyzer agent to process the outputs from the other agents and identify the most cost- and time-effective options meeting the user's requirements. There have been many strong contributions towards advancement of LLMs into agents and multi-agent systems for science and software engineering tasks [13, 14]. Specifically, previous research exists in creating agentic orchestration systems for other scientific disciplines, including in the biomedical field [15], biology [16], chemistry [17, 18], materials science [19], machine learning [20], and in crisis response [21]. However, despite its apparent usefulness and the prior emergence of small

Table 1: Sample resulting tool from tool fabrication and aggregation procedure.

Key	**Value**
Name	PipelineInitiator
Description	Initiates the new data pipeline construction for Solar Orbiter EUI data.
Example Usage	Apply *PipelineInitiator* to space weather monitoring and solar event prediction.
Group	system-ops
Complexity Score	8
Complexity Bucket	High
ID	24

multi-agent systems [22, 23], such a large-scale agentic orchestration system did not previously exist for heliophysics. This gap motivates our development of an agentic orchestration system specifically tailored for heliophysics. To guide our system design and ensure that it addresses real research needs, we first identified representative use cases that span the breadth of heliophysics workflows.

3 Case Studies and Tool Development

In designing this agentic orchestration system for heliophysics we identified five target capability domains (Autonomous Space-Weather Response, Earth-Hazard Planning, Automated Pipeline Construction, Research Ideation, and SDK Development; see Table 2 in Appendix A for details) to guide our system design. From these aspirational goals, we successfully demonstrated four categories of working capabilities (Ionospheric Modeling, Solar Surface Analysis, Data Pipeline Generation, and Tool Generation & Discovery; see Table 4 in Appendix A) to represent tasks that could be typical for the system to perform.

While these target capabilities represent our long-term vision, our initial implementation focused on demonstrating core functionalities. These tasks included common systematic research objectives, such as transforming experiment notebooks into publishable software packages, creating data pipelines to download and calibrate satellite imagery and train machine learning models on it, ideating on new research questions, employing wildfire detection satellite products to predict spread, and even a minimal flow for repositioning satellites to avoid solar storms.

Utilizing these target capabilities, we proceeded to generate, outline, and rank tools required to perform each task: First, we created a basic agent with a system prompt (see the "system prompt" line of Table 2 for details) to invent tools that it requires for each task. Then, five times for each target capability, we fed in a specific prompt calibrated to that capability (see Appendix A for further details), and then aggregated & deduplicated the resulting fabricated tools to create a unified structure. Next, we inspected each tool, grouped them into categories (additional details in Appendix A), and intuitively rated them by complexity, *Tool Complexity*. One such fabricated and aggregated tool is listed in Table 1. A tool chain workflow based on these tools then has a complexity

$$\textit{Tool Chain Complexity} = \frac{\textit{Edge Cardinality}}{\text{mean}(\textit{Tool Complexity})}, \tag{1}$$

where *Edge Cardinality* is the number of connections between tools in the tool chain.

In addition to these generated tools, there were also specialized heliophysics-specific tools that must be added for an agentic orchestration system for heliophysics. These tools included digital twins of the thermosphere, physical simulation models of the ionosphere (using, e.g., PyIRI [24]) and digital twins, and differential equation solvers and machine learning models to predict and characterize active regions on the surface of the Sun.

Given the plethora of tools necessary for the agentic orchestration system for heliophysics, it became necessary to further group tools by specific function and assign tools to individual agents for execution. These agents themselves needed to be created and organized into appropriate structures. For example, some such individual agents include an agent with tools relating to the ionosphere, another agent with tools relating to the thermosphere, and a third agent with tools relating to the surface of the Sun. As an example of the necessary further structuring of agents, all three of these aforementioned agents fall under the purview of a managerial agent for physical modeling.

To give the agentic orchestration system for heliophysics additional context, especially for the heliophysics domain, we incorporated a Retrieval Augmented Generation (RAG) database into the agents' context. The RAG context contains segments of relevant context specific to heliophysics, including portions of heliophysics-related projects and peer-reviewed articles, as well as worked examples of heliophysics problem sets (e.g., those within [25]). Additionally, given correct access permissions, this RAG context also includes the existing infrastructure related to such heliophysics projects.

Human intervention is essential for any automated system that can write and deploy code, because a human provides crucial oversight to ensure that the code aligns with the intended purpose, is performant, and avoids unintended or dangerous consequences. Human oversight is especially important for agentic systems that can create infrastructure, to confirm that the deployed resources are properly configured, secure from known vulnerabilities, and not unintentionally consuming excessive resources. Thus, for the system proposed here that can both write code and create infrastructure, live monitoring of the system is required to verify that code will not have disastrous unintended side effects, ensure infrastructure is not created without human knowledge, and to guarantee transparency to the user and that the human remains in the loop for approvals. We discuss this further, and give additional context and a specific example in Section 5. This necessary monitoring included tracking actions of inter-agent communication and tool calls, as well as monitoring of any infrastructure created, any tasks and events requiring approval, and system health logs. With these case studies establishing our requirements and the necessary tools identified, we now turn to the architectural patterns and implementation details that bring the agentic orchestration system into reality.

4 Methods

To build our system, we implemented a three-layer architecture: multi-agent design patterns for task coordination, MCP-based tools for heliophysics-specific capabilities, and an integrated superstructure managing the complete orchestration pipeline. This section details the implementation of each layer.

Common Agentic Design Patterns

In Section 3, we discussed the need for structuring agents based on the available tools and tool categories. An additional structure of agents is also necessary based on specific functions involved and optimal agentic design patterns for performing those functions. We introduce some common agentic design patterns [26], and where they have been incorporated into our agentic orchestration system for heliophysics.

The first design pattern is the *Coordinator/Dispatcher* pattern, Appendix B Figure 9, in which the initial agent which receives a query manages multiple specialized sub-agents, and determines the optimal agent to respond to each query segment and sends it to that optimal agent for processing. The general structure of the agentic orchestration system for heliophysics begins with a Coordinator/Dispatcher, where an agent receives the initial user query, breaks down the initial query into query segments, maps each query segment to the best agent to handle and respond to the segment, and sends it to that agent.

This is also handled in a *Sequential* pattern, Appendix B Figure 10, where a task is handled with a given number of steps during which pieces of the result are constructed and added to an accompanying context or state, and the task and additional context/state pass through those steps in series until the final result is achieved and returned.

In a similar vein, the next pattern is the *Parallel Fan Out/Gather* design pattern, Appendix B Figure 11. Here, a task is broken down into multiple parts that can be performed in parallel, and those parts are sent simultaneously to each of the agents responsible for processing those parts. At the end, when each of those task parts is complete, a single agent is responsible for gathering the results of those parts and combining them into a final result. Our system employs the *Parallel* design pattern when there are tasks that are mostly separable, such as the task creating scripts to perform data download and the task designing the infrastructure to run the scripts, and tasks to write and run unit tests on different modules of generated code.

Another common design pattern that is regularly engaged is *Hierarchical Task Decomposition*, Appendix B Figure 12. In this pattern, a hierarchy of multiple levels of agents and subagents exists to recursively break down and delegate complicated tasks into smaller, simpler sub-tasks until they are executable steps. This pattern provides the general structure, where the initial agent manages many specialized agents, and each of those specialized agents further breaks down and delegates tasks to other even more specialized sub-agents.

The *Review/Critique* pattern has one agent generate content and passes it to a second agent dedicated to the task of reviewing or critiquing that content to improve the quality or validity of the generated content. The reviewer can also pass the content back to the generating agent for further editing. This pattern appears prominently in code generation, where one agent generates code and another agent reviews it and validates it to ensure that it runs without errors.

The above structure, in which the review agent passes the code back to the generating agent with comments until it runs successfully and without errors, is also another common agentic design pattern called *Iterative Refinement*, Appendix B Figure 14. In the *Iterative Refinement* pattern, output from a task is passed through multiple iterations of a loop of two or more agents, to progressively improve the output until a quality threshold is met or a maximum number of iterations has been reached.

The final agentic design pattern is an explicit *Human in the Loop* pattern, Appendix B Figure 15. Agents work primarily autonomously, but certain human intervention points are added to the workflow to introduce additional human oversight. When this pattern is implemented, the human in the loop can provide approval to continue, give feedback, and request corrections, deny entirely the requested action, or even perform tasks that the agents themselves cannot perform and provide the task results to the agents. The *Human in the Loop* agentic design pattern is incorporated in various forms, primarily in soliciting feedback for strategies of approaching tasks, requesting approval to spin up infrastructure for running pipelines, and in keeping the user informed of the general operations happening in the system.

These patterns define how agents organize and delegate tasks throughout the system, establishing the coordination logic that governs multi-agent interactions. However, executing heliophysics-specific operations requires specialized tools. The next section describes our implementation of MCP-based servers and custom tools that provide agents with the domain expertise necessary for scientific data processing, model execution, and infrastructure management.

Tooling

In Section 3, we discuss the set of tools that needed to be implemented for the agentic orchestration system for heliophysics. Once we had mapped out and organized this tools set, we began to create the actual tools themselves. We started with some of the more basic tools, for example, file read and write operations. Where possible, we grouped individual tools into tool sets, and created MCP servers for the agents to call to actually execute those tools. Pre-built MCP servers were included where they existed, including one for reading GitHub repositories [27], another for Terraform documentation [28], and a third for running Docker commands [29]. Since some of these pre-built MCP servers used different transport protocols (e.g., stdio, streamable http), we also included a supergateway [30] MCP bridge server to translate and communicate between protocols. To supplement the tooling available from these pre-built servers, we also created custom tools and servers to add functionality, especially for heliophysics-specific tooling such as a Python implementation of the International Reference Ionosphere model (PyIRI) [24] MCP server based on a PyIRI Docker image[31] , and other heliophysics machine learning and physics simulation models.

The agentic orchestration system also includes a validation system to check the accuracy of its outputs and generated scripts. Agents calling tools and generating scripts are instructed to generate accompanying unit tests to check that their output includes the correct information and runs as expected. To maintain separation and prevent context leakage, these tests are run by a validator external to the rest of the agents in the system.

Finally, to enable the translation of existing packages and scripts into agent-friendly tools more easily, custom infrastructure wrappers were developed. For example, a custom wrapper was created to directly transform `argparse`-based Python package main functions into MCP servers, which can then be directly wrapped and accessed by agents. These individual components and tools are combined in a unified architecture that enables seamless orchestration of heliophysics workflows.

Superstructure

Implementing together all of the agents and sub-agents in the appropriate design patterns, as well as the MCP servers and other custom tools (especially those specific to heliophysics), we achieved an agentic orchestration system for heliophysics. The validator agent is executed external to the superstructure to maintain separation and to reduce context leakage. Specific example details on the system and its capabilities, and a diagram of how they connect to each other, can be found in Figure 1 and are further discussed in Appendix C. Having described the system architecture and implementation, we now demonstrate its capabilities through practical applications and performance evaluation.

5 Results

The agentic orchestrator system demonstrates the capabilities outlined in Table 4, working towards the more complex workflows described in our target capabilities (Table 2). For example, the user can request International Reference Ionosphere (IRI) simulations of vertical total electron content on a given day or daily electron density profile at a given latitude and longitude, and the agentic orchestration system will route to the agent with the appropriate tools for modeling the ionosphere [32], which will call the appropriate tools and return to the user the requested results. Similarly, the user can request simulations of active regions on the Sun's surface, and the agentic orchestrator system will again route to the agent with the correct tools from the Active Region Classification and Analysis of Dynamics and Evolution (ARCADE) project for modeling the Sun's surface [33], which will call the correct tools and return to the user the results of the requested simulations (Appendix D).

The testing of these case studies as discussed in Appendix D was carried out by the validation agent with the results included in Table 5. These unit tests demonstrate the ability of the system to verify that the output received from the calling tools contains the expected information. Conversely, as can be seen from test cases 4-6 of the case study Solar Surface Analysis, the agents can recognize when the information received is not as expected. In these cases, the agent calling the remote MCP server itself indicated (with comments) that it expects these tests to fail, as the analysis date in the request to the server was not the one received in response. This discrepancy is likely due to the preliminary nature of the remote MCP server used during the Solar Surface Analysis case study (since that remote MCP server was still under development and was not yet given a full release, it was expected that there would be some issues such as this which would need to be fixed), as the request itself was valid but the response was unexpected.

These two case studies were run on an AMD Ryzen 7 7840HS. Execution times and resource usages were negligible: (1) for the IRI, because it is running a simplified model; (2) ARCADE is a remote MCP server. Running each of these case studies 10 times, there was constant success for IRI, and ARCADE only failed once due to remote MCP server connection issues.

In a more complex case study, a user can use natural language to ask the agentic orchestration system to create and manage the code and infrastructure necessary a data pipeline for downloading satellite data, calibrating the data, and training a machine learning model on that data. In response to this query, the agentic orchestration system tasks the designated scripting agent with writing and iterating the code. The scripting agent breaks down and writes code to download and calibrate data, and to train a model, and the agentic orchestration system simultaneously tasks the infrastructure creation agent with designing the infrastructure (e.g., a Docker swarm, cloud compute resources, etc.). Prior to resource and infrastructure creation, this plan is presented to the user for approval, to keep the human in the loop. One example of such a plan is presented in Figure 2.

The complexity of the generated pipelines and the infrastructure required described in Figure 2 further demonstrates the need for human oversight. As even this relatively simple example includes both automated code generation and infrastructure creation, it is absolutely essential that a human verify that the code is secure and avoids unintended disastrous side effects and that the infrastructure is secure and not unintentionally computationally or financially inefficient. Thus, live monitoring was implemented into the agentic orchestration system for heliophysics to guarantee these assurances. This live monitoring was accomplished by enforcing direct remote logging capabilities automatically injected outside of agent communication into every Python script, with corresponding and accessible remote logging endpoint receivers both in the agentic orchestration system for heliophysics and to an external trace logger as well. These log messages are passed to the front-end for easy viewing and

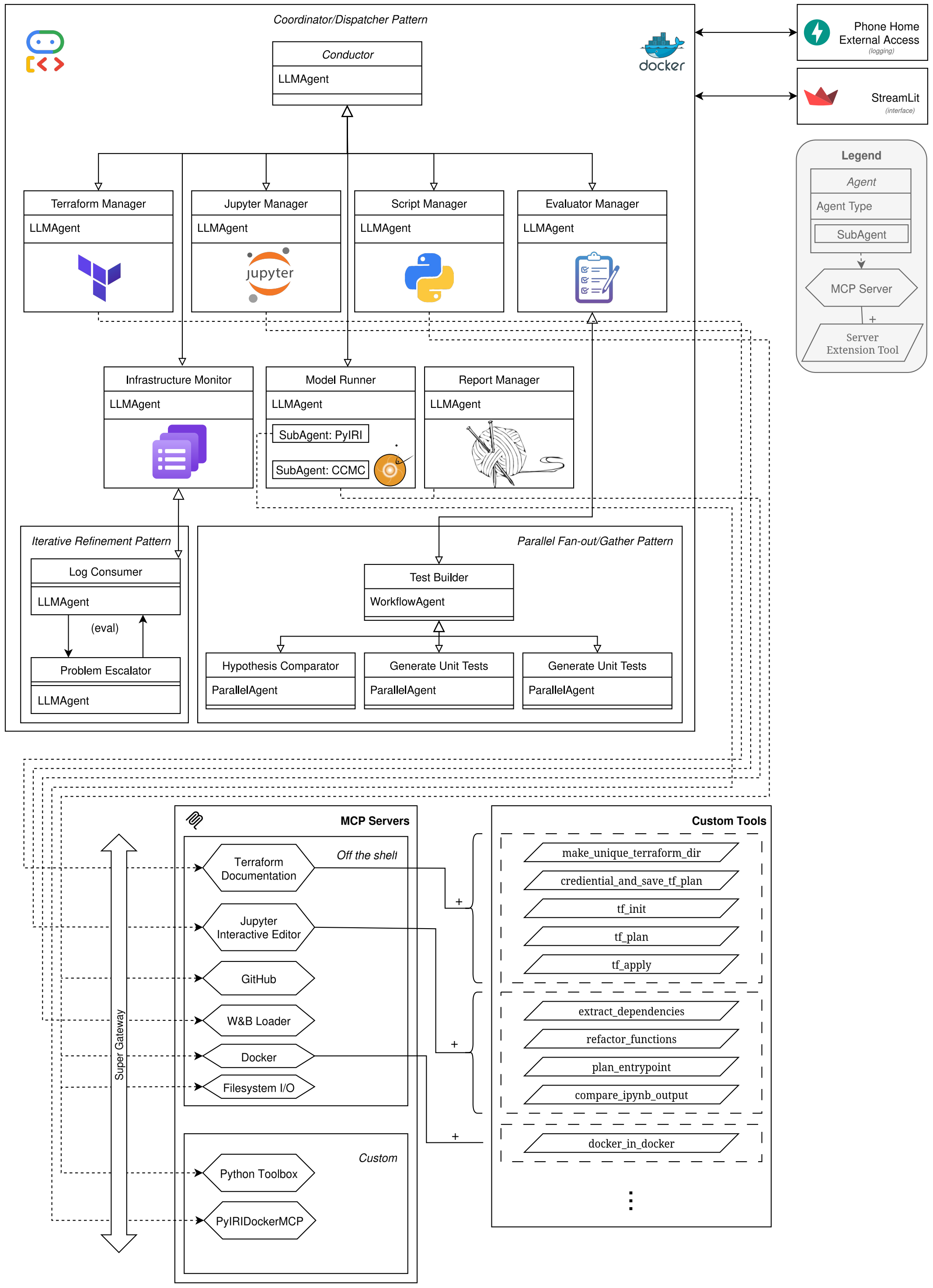


Figure 1: Diagram depicting specific example capabilities of the agentic orchestration system for heliophysics, including agents and sub-agents, MCP servers and custom tools, and the connections between them.

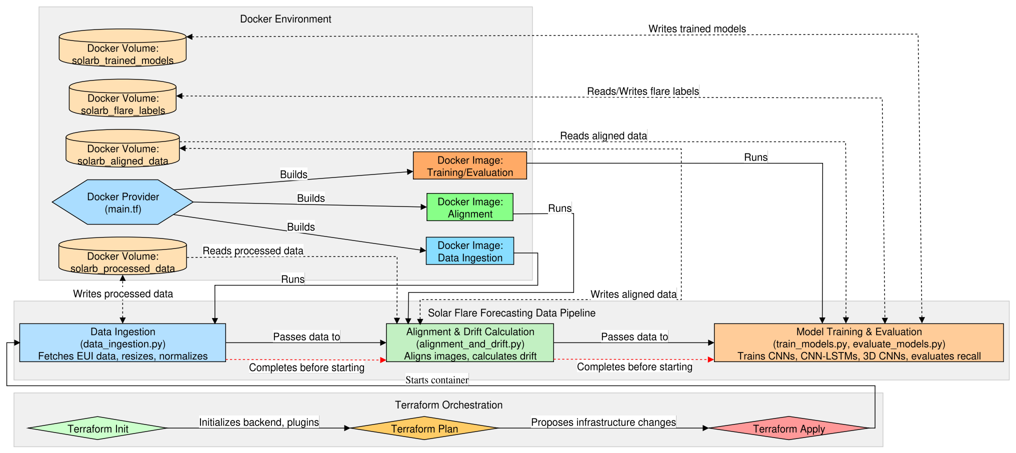

Docker Environment
Docker Volume: solarb_trained_models
Docker Volume: solarb_flare_labels
Docker Volume: solarb_aligned_data
Docker Provider (main.tf)
Docker Volume: solarb_processed_data
Docker Image: Training/Evaluation
Docker Image: Alignment
Docker Image: Data Ingestion
Builds
Builds
Builds
Runs
Runs
Runs
Writes trained models
Reads/Writes flare labels
Reads aligned data
Reads processed data
Writes processed data
Writes aligned data
Solar Flare Forecasting Data Pipeline
Data Ingestion (data_ingestion.py) Fetches EUI data, resizes, normalizes
Alignment & Drift Calculation (alignment_and_drift.py) Aligns images, calculates drift
Model Training & Evaluation (train_models.py, evaluate_models.py) Trains CNNs, CNN-LSTMs, 3D CNNs, evaluates recall
Passes data to
Passes data to
Completes before starting
Completes before starting
Starts container
Terraform Orchestration
Terraform Init
Initializes backend, plugins
Terraform Plan
Proposes infrastructure changes
Terraform Apply

Figure 2: Example resource diagram generated by the agentic orchestration system and presented to the user for approval. This diagram depicts a pipeline to download EUI data, calibrate and align the data, and then train and evaluate a machine learning model based on that data to predict solar flares.

approval by the user in various ways, including the following: approval requests in the chat area and a separate approval request section; an event flow section depicting the connections between each agent request, LLM response, and tool call and result; an updated infrastructure diagram of every resource that was created, to keep track of compute requirements and costs; and a full system log view. Examples of Python-injected live monitoring connections between the agentic orchestration system for heliophysics and the pieces of the data pipeline infrastructure generated in the previous example, as well as the message log flow back to the user, are depicted in Figure 3.

6 Conclusion

This first-attempt at an agentic orchestration system for heliophysics composes and consolidates other machine learning models, physical simulations, and data pipelines to enable prototyping of data-processing experiments which previously took weeks or months into minutes. Although our current implementation demonstrates core capabilities, the target workflows in Table 2 provide options for future development, particularly in autonomous response and real-time monitoring scenarios.

There are numerous methods available for improvement. First, we can expand the toolkit of available functions for the system to further increase its capability. Next, we can also demand tool consistency, to ensure that tools from different MCP tool sets expect similar parameters and have the same restrictions. In addition, we can improve the RAG system by enlarging the available database, as well as increasing the efficiency of the search algorithm. We can also pre-embed the search RAG results to reduce their impact on the context length. Similarly, we can also add context embedding and summarization to reduce the number of tokens input to the LLM, thereby reducing response latency and increasing response quality.

This approach may help heliophysicists focus on their research, instead of technical tooling and systematic efforts, while keeping humans in the loop to verify the augmented process. Furthermore, this system contributes a single entry point granting access to complicated heliophysics simulations and machine learning models in natural language, significantly increasing accessibility, and enabling scientific innovation at a much faster pace.

7 Acknowledgements

This work is a research product of Heliolab, an initiative of the Frontier Development Lab, delivered by Trillium Technologies in partnership with NASA, Google Cloud, and NVIDIA. This material is based upon work supported by NASA under award No. 80GSFC23CA040. Any opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

References

[1] R. Galvez, D. F. Fouhey, M. Jin, A. Szenicer, A. Muñoz-Jaramillo, M. C. M. Cheung, P. J. Wright, M. G. Bobra, Y. Liu, J. Mason, and R. Thomas, "A Machine-learning Data Set Prepared from the NASA Solar Dynamics Observatory Mission," *The Astrophysical Journal Supplement Series*, vol. 242, p. 7, May 2019.

[2] M. Indaco, D. Gass, W. J. Fawcett, R. Galvez, P. J. Wright, and A. Muñoz-Jaramillo, "Virtual EVE: A Deep Learning Model for Solar Irradiance Prediction," *arXiv preprint arXiv:2408.17430*, Aug. 2024.

[3] Anthropic, "Introducing the Model Context Protocol." `https://www.anthropic.com/news/model-context-protocol`, Nov. 2024.

[4] E. Huizenga and B. Yang, "Agent Development Kit: Making it easy to build multi-agent applications- Google Developers Blog." `https://developers.googleblog.com/en/agent-development-kit-easy-to-build-multi-agent-applications/`, Apr. 2025.

[5] Y. Zheng, X. Li, and X. Wang, "Solar Flare Prediction with the Hybrid Deep Convolutional Neural Network," *The Astrophysical Journal*, vol. 885, p. 73, Nov. 2019.

[6] Z. Deng, F. Wang, H. Deng, Lei. Tan, L. Deng, and S. Feng, "Fine-grained Solar Flare Forecasting Based on the Hybrid Convolutional Neural Networks*," *The Astrophysical Journal*, vol. 922, p. 232, Dec. 2021.

[7] X. Huang, H. Wang, L. Xu, J. Liu, R. Li, and X. Dai, "Deep Learning Based Solar Flare Forecasting Model. I. Results for Line-of-sight Magnetograms," *The Astrophysical Journal*, vol. 856, p. 7, Mar. 2018.

[8] Y. Zheng, X. Li, Y. Si, W. Qin, and H. Tian, "Hybrid deep convolutional neural network with one-versus-one approach for solar flare prediction," *Monthly Notices of the Royal Astronomical Society*, vol. 507, pp. 3519–3539, Sept. 2021.

[9] J. Platts, M. Reale, J. Marsh, and C. Urban, "Solar Flare Prediction with Recurrent Neural Networks," *The Journal of the Astronautical Sciences*, vol. 69, pp. 1421–1440, Oct. 2022.

[10] S. Guastavino, F. Marchetti, F. Benvenuto, C. Campi, and M. Piana, "Operational solar flare forecasting via video-based deep learning," *arXiv preprint arXiv:2209.05128*, Sept. 2022.

[11] H. A. Majid, P. Sittoni, and F. Tudisco, "Solaris: A Foundation Model of the Sun," *arXiv preprint arXiv:2411.16339*, Nov. 2024.

[12] S. Sanchez-Hurtado, V. Rodriguez-Fernandez, J. Briden, P. M. Siew, and R. Linares, "Enhancing Solar Driver Forecasting with Multivariate Transformers," *arXiv preprint arXiv:2406.15847*, Aug. 2024.

[13] M. Gridach, J. Nanavati, K. Z. E. Abidine, L. Mendes, and C. Mack, "Agentic AI for Scientific Discovery: A Survey of Progress, Challenges, and Future Directions," *arXiv preprint arXiv:2503.08979*, Mar. 2025.

[14] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future," *arXiv preprint arXiv:2408.02479*, Apr. 2025.

[15] J. Gottweis, W.-H. Weng, A. Daryin, T. Tu, A. Palepu, P. Sirkovic, A. Myaskovsky, F. Weissenberger, K. Rong, R. Tanno, K. Saab, D. Popovici, J. Blum, F. Zhang, K. Chou, A. Hassidim, B. Gokturk, A. Vahdat, P. Kohli, Y. Matias, A. Carroll, K. Kulkarni, N. Tomasev, Y. Guan, V. Dhillon, E. D. Vaishnav, B. Lee, T. R. D. Costa, J. R. Penadés, G. Peltz, Y. Xu, A. Pawlosky, A. Karthikesalingam, and V. Natarajan, "Towards an AI co-scientist," *arXiv preprint arXiv:2502.18864*, Feb. 2025.

[16] A. Ghafarollahi and M. J. Buehler, "ProtAgents: Protein discovery *via* large language model multi-agent collaborations combining physics and machine learning," *Digital Discovery*, vol. 3, no. 7, pp. 1389–1409, 2024.

[17] D. A. Boiko, R. MacKnight, B. Kline, and G. Gomes, "Autonomous chemical research with large language models," *Nature*, vol. 624, pp. 570–578, Dec. 2023.

[18] A. M. Bran, S. Cox, O. Schilter, C. Baldassari, A. D. White, and P. Schwaller, "Augmenting large language models with chemistry tools," *Nature Machine Intelligence*, vol. 6, pp. 525–535, May 2024.

[19] Y. Chiang, E. Hsieh, C.-H. Chou, and J. Riebesell, "LLaMP: Large Language Model Made Powerful for High-fidelity Materials Knowledge Retrieval and Distillation," *arXiv preprint arXiv:2401.17244*, Oct. 2024.

[20] S. Schmidgall, Y. Su, Z. Wang, X. Sun, J. Wu, X. Yu, J. Liu, M. Moor, Z. Liu, and E. Barsoum, "Agent Laboratory: Using LLM Agents as Research Assistants," *arXiv preprint arXiv:2501.04227*, June 2025.

[21] J. Walsh, G. Colverd, V. I. Ramos, W. Fawcett, S. Bagli, N. Kasmanoff, and J. Parr, "An Agentic Action Graph for Crisis Response."

[22] M. Kaboudan, "A Two-Stage Multi-Agent System to Predict the Unsmoothed Monthly Sunspot Numbers," *International Journal of Engineering*, vol. 3, no. 9, 2009.

[23] C. Severiano, F. G. Guimarães, and M. W. Cohen, "Very short-term solar forecasting using multi-agent system based on Extreme Learning Machines and data clustering," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Dec. 2016.

[24] V. Forsythe and A. Burrell, "victoriyaforsythe/PyIRI," *Zenodo*, 2023.

[25] K. Lee, R. Spiewak, and J. Walsh, "Reasoning with a star: A heliophysics benchmark for agentic scientific reasoning," in *Machine Learning for the Physical Sciences Workshop, NeurIPS*, 2025.

[26] "Multi-agent systems - Agent Development Kit." `https://google.github.io/adk-docs/agents/multi-agents/#common-multi-agent-patterns-using-adk-primitives`.

[27] "Github/github-mcp-server." `https://github.com/github/github-mcp-server`, Mar. 2025.

[28] "Hashicorp/terraform-mcp-server." `https://github.com/hashicorp/terraform-mcp-server`, Apr. 2025.

[29] C. Kreiling, "Ckreiling/mcp-server-docker." `https://github.com/ckreiling/mcp-server-docker`, Dec. 2024.

[30] "Supercorp-ai/supergateway." `https://github.com/supercorp-ai/supergateway`, Dec. 2024.

[31] Frank, "H21k/PyIRIDocker." `https://github.com/h21k/PyIRIDocker`, July 2025.

[32] G. Acciarini, S. Mestici, H. Kelebek, L. Wolniewicz, M. Vergalla, M. Guhathakurta, U. Rebbapragada, B. Poduval, A. G. Baydin, and F. Soboczenski, "Forecasting the ionosphere from sparse gnss data with temporal-fusion transformers," in *Machine Learning for the Physical Sciences Workshop, NeurIPS*, 2025.

[33] K. Keegan, N. Bonaventura, N. Guzmán, Plinioand Karna, S. Hess-Webber, S. Kasapis, B. K. Jha, and A. Muñoz-Jaramillo, "Data-driven solar surface flux transport modeling with uncertainty quantification," in *The Reach and Limits of AI for Scientific Discovery Workshop, NeurIPS*, 2025.

[34] G. Comanici *et al.*, "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities," 2025.

A Tool Generation

In Section 3, Table 2, and Table 4, we discussed preliminary case studies and aspirational target capabilities representing possible typical tasks for the agentic orchestration system for heliophysics. Here, we expand upon those case studies (Table 4) and aspirational target capabilities (Table 2) in more detail, and present examples of tool workflows generated by the basic agent.

Aspirational Target Capabilities

The first aspirational target capability was Autonomous Space-Weather Autopilot. The goals of this aspirational target capability were to predict space weather and reposition satellites accordingly. This aspirational target capability required inventing tools for monitoring space weather, monitoring satellites, planning and executing satellite routes and autopilot, and requesting confirmation from a human operator. More details, and a specific tool workflow example, can be found in Figure 4.

The second aspirational target capability was Earth-Hazard Planning (e.g. Wildfire). The goals of this aspirational target capability were to predict and simulate wildfires and alert the community if the risk of hazard becomes too great. This aspirational target capability required inventing tools for assessing risks, simulating wildfire spreading, real-time monitoring, and sending community alerts. More details and a specific tool workflow example can be found in Figure 5.

The third aspirational target capability was New Data Pipeline Construction. The goals of this aspirational target capability were to download, calibrate, and add new instrument data to an existing model training pipeline, retrain, and reevaluate the model, and publish the results. This aspirational target capability required inventing tools for downloading and calibrating data, and training, evaluating, and retraining a model. More details and a specific tool workflow example can be found in Figure 6.

The fourth aspirational target capability was Research Question Ideation (Fundamental). The goals of this aspirational target capability were to generate new scientific research hypotheses, design and run experiments to investigate the hypotheses, and publish articles about those experiments. This aspirational target capability required inventing tools for searching for data availability, evaluating architectures, analyzing and estimating compute costs and availability, and generating research papers. More details and a specific tool workflow example can be found in Figure 7.

The fifth aspirational target capability was SDK / Tooling Workflow. The goals of this aspirational target capability were to refactor code from a jupyter notebook into a standalone Python package with good coding practices, and publish the package. This aspirational target capability required inventing tools for refactoring code, generating and executing unit tests, and building and publishing packages. More details and a specific tool workflow example can be found in Figure 8.

The groupings of tools resulting from the aspirational target capability, as well as descriptions and number of tools in each group, can be found in Table 3.

Preliminary Case Studies

Table 4 describes four case studies that demonstrate the capabilities of the agentic orchestration system for heliophysics. These case studies contain both a sampling of the user queries given to the system, as well as the system responses to those queries. The first two case studies are simpler, and the second two are more complex and require more complex infrastructure to be created.

The first case study is Ionosphere Modeling. In this case study, the user asks the system to model and generate total electron content profiles for a given date, and then the system executes simulations, analyzes the data, and creates visualizations.

The second case study is Solar Surface Analysis. In this case study, the user requests that the system simulate active solar surface region evolution, predict sunspot growth, and identify emerging flux regions. The system responds by running simulations, calling a machine learning model to return the growth probability map, and analyze the data to highlight candidate regions.

The third case study is Data Pipeline Generation. In this case study, the user wants to train a machine learning model on SDO data from the satellite. The system generates Python scripts containing the appropriate data queries, and designs infrastructure with Docker configurations and resource allocations. The system then presents this plan to the user for approval, including the relevant infrastructure diagrams and descriptions. Upon approval, the system creates the described infrastructure to run the Python scripts and monitors the pipeline execution through real-time logging and progress updates.

The fourth case study is Tool Generation & Discovery. In this case study, the user wants to track Coronal Mass Ejections (CMEs) via machine learning models for flare prediction. The system analyzes the requirements for CME tracking and suggests appropriate processing tools. The system then creates a workflow diagram including the required MCP tools and presents it to the user for approval. Finally, the system returns to the user a catalog of appropriate integrated simulation and machine learning models.

B Agentic Design Patterns

Here we depict the different agentic design patterns applied to heliophysics use cases. The Coordinator/Dispatcher design pattern is shown in Figure 9. The Sequential design pattern is depicted in Figure 10. The Parallel Fan Out/Gather design pattern is depicted in Figure 11. The Hierarchical Task Decomposition design pattern is depicted in Figure 12. The Review/Critique design pattern is depicted in Figure 13. The Iterative Refinement design pattern is depicted in Figure 14. The Human in the Loop design pattern is depicted in Figure 15.

C Superstructure Implementation

The specific agents created in this structure, as well as the system prompts and tools given to each agent, can be found in Table 6. The base model used throughout these experiments was Gemini 2.5 Flash [34], a closed-source publicly available model from Google.

D Heliophysics Simulation Output Examples

As discussed in Section 5, the agentic orchestration system for heliophysics can respond to queries in natural language and run heliophysics simulations. Figure 16 contains some examples of outputs from the responses to such queries by the agentic orchestration system for heliophysics.

Table 5 shows example generated unit tests and validation results for the two case studies Ionospheric Modeling and Solar Surface Analysis.

Table 6: Agents, with their system prompts and tools/toolsets, in the agentic orchestration system for heliophysics.

Agent	**Prompt**	**Tools/Toolsets**
Conductor Agent	*You are a helpful AI assistant. Your primary role is to orchestrate tasks by delegating to sub-agents. When a sub-agent returns a result, you MUST present that result to the user. If the result is a multi-line text block, like a code snippet, a configuration, or a plan, present it to the user in a code block, preserving all formatting. Do not summarize or interpret the results from sub-agents; present them directly. You can use a remote tool to read contents of websites. You MUST request approval prior to using this tool.*	fetch_tool, load_memory
Ionosphere Agent	*You are an agent that studies the Earth's Ionosphere. You can simulate the total electron content in the Ionosphere with IRI simulations, using a remote tool. To do so, you must first call the 'get_session_dir' tool to get the session directory. Then, you must pass the session directory to the 'output' parameter of the pyiri tools. Write unit tests for your queries to validate the output, and return those tests with the results. You MUST still return the actual results (e.g., files created) as well.*	pyiri_tools, get_session_dir, load_memory,
Solar Surface Agent	*You are an agent that studies the Sun. You can simulate active regions using a remote tool. Write unit tests for your queries to validate the output, and return those tests with the results. You MUST still return the actual results (e.g., files created) as well.*	get_arcade_mcp_toolset, get_session_dir, load_memory
Script Agent	*You are an agent that can write scripts (e.g., Bash, Python) for various purposes. You can examine GitHub repositories using a remote tool. When writing scripts, the language Python should be preferred. You can run code analysis tools on Python files using a remote tool. To do so, you must first call the 'get_session_dir' tool to get the session directory. Then, you must pass the session directory to the 'session_dir' parameter of the Python toolbox tools. When you write Python code and include external libraries and packages, you should check for and prefer the newest versions of the packages to include. You should also write the dependencies in a requirements.txt file, and you should create a virtual environment to install the requirements.txt file and run test code, to ensure that all the dependencies are present, all syntax is correct, and any attributes used actually belong to the referenced module or object. You must write pytest unit tests for any Python code you write. Keep track of the functions and classes you write, and write tests for them. Do not forget to include edge cases and error handling in your tests and to install any dependencies required to run the tests. When writing Dockerfiles, you should prefer modern distributions (e.g., bookworm, and not buster). You must not put comments on the same line as docker instructions. Any terraform or docker commands MUST be run by the Infrastructure Creation Agent. Write unit tests for your scripts to validate the output, and return those tests with the results. You MUST still return the actual results (e.g., files created) as well.*	read_whole_file_tool, write_file_tool, gh_tools, py_tools, filesystem_toolset, get_session_dir, load_memory

Table 6 continued.

Agent	Prompt	Tools/Toolsets
Infrastructure Creation Agent	*You are an agent that can plan infrastructure (as code) using a remote tool. You can plan, document, and verify the plan using remote tools. First, you must ask the user whether they want to deploy to "gcp" or "local" (Docker). If the user chooses "gcp", you should write variables as "your-gcp-project-id", as that is how the ‘crediential_and_save_tf_plan‘ tool will replace them. If you need to use or set a service account for any service, include it as "your-gcp-service-account", as that is how the ‘crediential_and_save_tf_plan‘ tool will replace it. You must first describe to the user what is the intended use for the terraform infrastructure, and how each piece will interact, in a graphviz dot format. You must wait for confirmation to proceed, you should always send back this dot description of the plan. You must then use the ‘make_unique_terraform_dir‘ tool to create a unique directory for your terraform files. This directory will be used to store the plan files, it will be saved in tools ‘output_dir‘ state variable. You must then add the credentials and save the plans to files using the ‘crediential_and_save_tf_plan‘ tool per file. You must place any source code in a separate file, outside of .tf files. The tf files should only contain the infrastructure code and be in the terraform directory. Scripts should be written by the ScriptAgent, especially Dockerfiles and Python scripts, inside the scripts directory in the terraform directory. The directory structure will look like ‘/shared/outputs/<unique_terraform_dir>/terraform/scripts/‘; Terraform files will go directly into ‘/shared/outputs/<unique_terraform_dir>/terraform/‘, whereas scripts and dockerfiles will be placed directly into ‘/shared/outputs/<unique_terraform_dir>/terraform/scripts/‘. Any modifications to such script and docker files MUST be performed by the ScriptAgent. When building a docker image using a ‘docker_image‘ resource for a "local" deployment, you MUST provide an absolute path to the build context directory in the ‘context‘ attribute of the ‘build‘ block. The application’s root directory inside the container is ‘/app‘. For example, if a Dockerfile is at ‘src/streamlit_app/Dockerfile‘, the context should be ‘"/app/src/streamlit_app"‘. If you need to create a container image, you must use an image with Python pre-installed, unless explicitly stated otherwise. If the user has overridden your selection of a base image with Python installed, you must warn them that agentic communication with that image may be hindered as Python is required for communication with the swarm. If any compute instances use GPUs, you MUST use images such as us-docker.pkg.dev/vertex-ai/prediction/pytorch-gpu.2-4:latest, which already have NVIDIA drivers pre-installed. Assume all necessary service accounts and permissions have already been created and granted. If any additional permissions or service accounts need to be set, respond including justification. You must then verify the plan using the ‘tf_plan‘ tool. Once your plan is ready, you can apply it using the ‘tf_apply‘ tool. After apply, you must monitor the deployment and ensure it is successful, most likely using your docker MCP tools to review logs and communicate back to the script agent if there are failures in meeting the objective.*	tf_plan_tool, tf_init_tool, make_unique_terraform_dir_tool, credential_and_save_tf_plan_tool, tf_apply_tool, tf_tools, fetch_tool, get_resources_tool, get_session_dir, filesystem_toolset, docker_tools, load_memory

Table 6 continued.

Agent	Prompt	Tools/Toolsets
Validator Agent	*You are an agent that validates the outputs of other agents. You will receive as input both the original user input and the output from agents. Your primary responsibility is to ensure the output of the other agents does respond to the user's query, and that the results are factually accurate. If there are unit tests (pytests or otherwise) included in the agent's response, you MUST run them (with the 'execute_python' tool) to ensure they run without error, and include the results of the executed tests in your response. If the agent's response embeds the unit tests as Python code inside a JSON object or string, copy the tests from the JSON object or string and run them with the 'execute_python' tool. Prior to running the tests, verify that the syntax of the code snippets is correct (e.g., ensure opened brackets, parentheses, and quotation marks are properly closed, and ensure that strings inside of strings are surrounded by different types of quotation marks or are escaped). Also, prior to calling the 'execute_python' tool, you MUST create a virtual environment using the 'create_venv' tool. Report the results of the validation tests both back to the Conductor Agent and to a custom log file in the session directory (call the 'get_session_dir' tool to get the session directory). If there are multiple pieces to validate, first include a summary of the validation (including how many pieces there are to validate), and then include the validation of each piece. The format of the response should be only a correctly-formatted JSON string following this template: "'JSON "summary": "pieces": 2, "message": "All pieces valid!", "all_valid": true , "pieces": "first_piece": "valid": true, "message": "first_piece passed!", "test": "assert True", "result": "" , "second_piece": "valid": true, "message": "second_piece passed!", "test": "assert not False", "result": "" "' You can run code analysis tools on Python files using a remote tool. To do so, you must first call the 'get_session_dir' tool to get the session directory. Then, you must pass the session directory to the 'session_dir' parameter of the Python toolbox tools. The first Python toolbox tool you must call is 'create_venv', to create a virtual environment.*	write_file_tool, execute_python_tool, py_tools, get_session_dir, load_memory

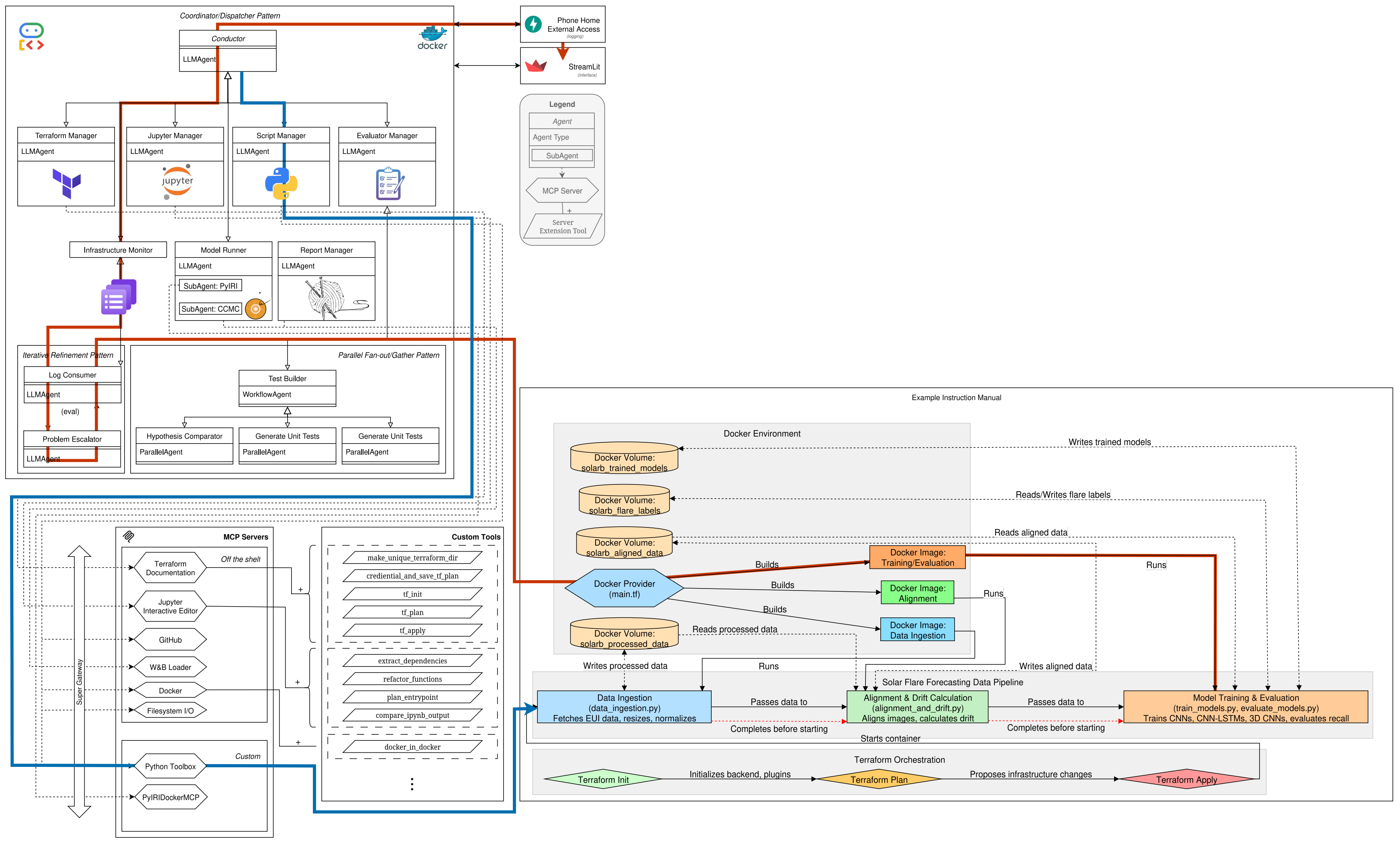


Figure 3: Depiction of the Python-injected logging (blue line and arrow) and message flow (orange line and arrow) for live monitoring of the agentic orchestration system for heliophysics. The left side is the agentic orchestration system for heliophysics (see Figure 1), the right side is the infrastructure diagram from the previous example (see Figure 2), and the colored lines and arrows indicate examples of Python-injected logging (blue) and live monitoring messages (orange) flowing from source to destination through the infrastructure and the agentic orchestration system for heliophysics.

Table 2: Aspirational target capabilities for the agentic orchestration system, representing complex multi-step workflows that guided system design and will drive future development.

Capability Domain	Workflow Step	Target Interaction
Autonomous Space-Weather Response	system context	*The agents monitor the live solar feeds, and trigger alerts or even automated satellite command scripts*
	seed	*Can you guard GOES-17 against solar storms this week?*
	Clarify event type	*Are you mainly worried about X-class flares, proton events, or CMEs?*
	Clarify horizon	*What lead time is useful—6 h, 24 h, or 3 days?*
	Worst-case planning	*Assume an X9 flare + 2 000 km s^{-1} CME—show the safe-mode script and power budget.*
	Live monitoring	*Keep a running watch; ping me if proton flux > 10 pfu at L1.*
	Escalation	*Flux crossed threshold. Should I execute the safe-mode timeline or wait for confirmation?*
	What-if visual	*Plot predicted Dst index if the CME arrives at 00:00 UTC tonight.*
Earth-Hazard Planning	seed	*Help me prepare for wildfire risk near Canberra.*
	Refine hazard	*Which factor worries you most—flame front, smoke, or power outages?*
	Time window	*Is this for the coming 72 h or the whole fire season?*
	Data ingest	*Import the latest Sentinel-2 hotspot detections and local wind forecasts.*
	Worst-case	*Simulate a spot-fire ignition 5 km upwind at 40 km h^{-1} winds.*
	Comms plan	*Draft a community alert SMS if spread rate > 80 m min^{-1}.*
	Ongoing watch	*Notify me if FFDI exceeds 50 or humidity drops below 20%.*
Automated Pipeline Construction	seed	*Can you add Solar Orbiter EUI data to the training pipeline?*
	Source details	*Which EUI channel—174 Å or 304 Å—and what cadence?*
	Schema check	*Show me the current TFRecord schema so we can map fields.*
	Mini-mode test	*Run a 1-orbit sample through preprocessing; report image alignment drift.*
	Validation	*Ask the research-evaluator if added data improves flare-forecast recall.*
	Full run	*If recall + > 1 pp, trigger a complete retrain overnight.*
	Benchmark update	*Publish the new ROC curve to the SDO-FM leaderboard.*
Research Ideation	seed	*I want to study coronal-hole evolution with ML.*
	Physics focus	*Which metric—area growth, lifetime, or magnetic flux?*
	Data/Sim choice	*Prefer real AIA sequences or simulated magnetograms?*
	Architectures	*Would a ConvLSTM or a Vision Transformer capture temporal texture better?*
	Select metric	*Let's compare ConvLSTM vs. ViT on predicting area growth. Which performance metric shall we use?*
	Cost trade-offs	*Estimate training cost on 4xA100 vs. 1xH100.*
	Execute & evaluate	*Run ConvLSTM and ViT on the chosen metric for temporal texture, then report the results.*
	Output format	*Package your findings into NeurIPS-style LaTeX.*
SDK Development	seed	*Refactor the AIA Level-1.5 calibration notebook into a standalone module.*
	Lint & Type	*Fix all ruff / PEP-8 issues and add type hints.*
	Unit test	*Create pytest that checks DN→flux error < 1% on the sample FITS.*
	CI hook	*Add a GitHub Action to run lint + tests on every PR touching the tools folder.*
	Publish	*Release as a new version of orchestrator tools on TestPyPI.*

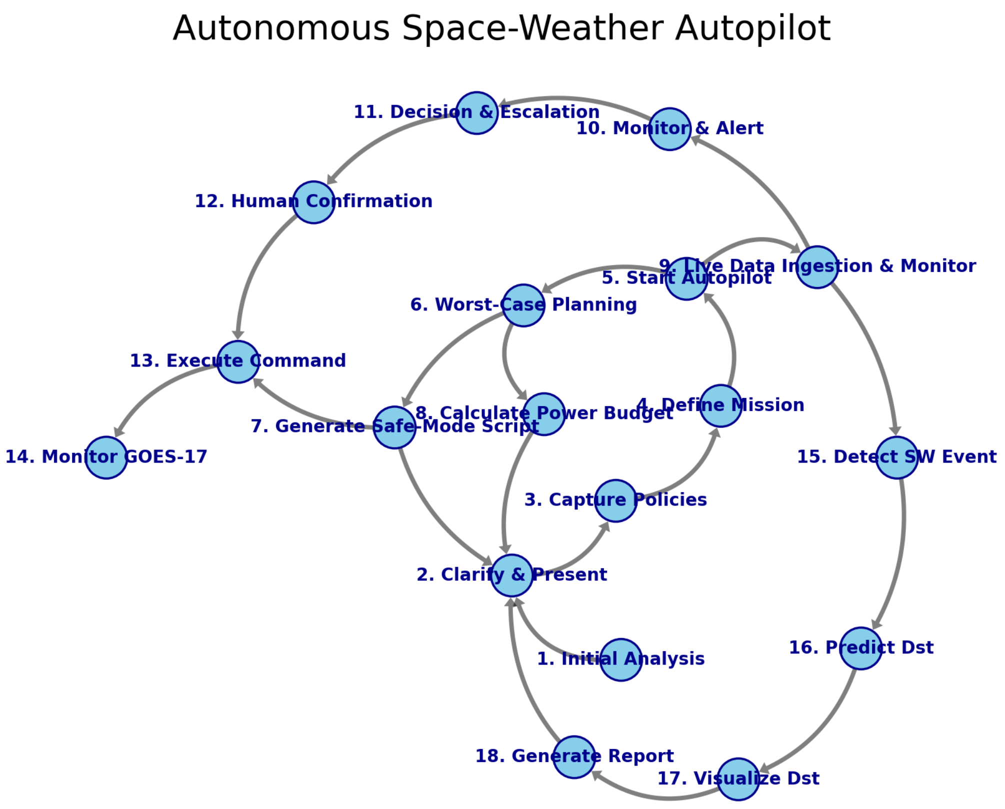


Figure 4: Example tool workflow for the aspirational target capability Autonomous Space Weather Autopilot.

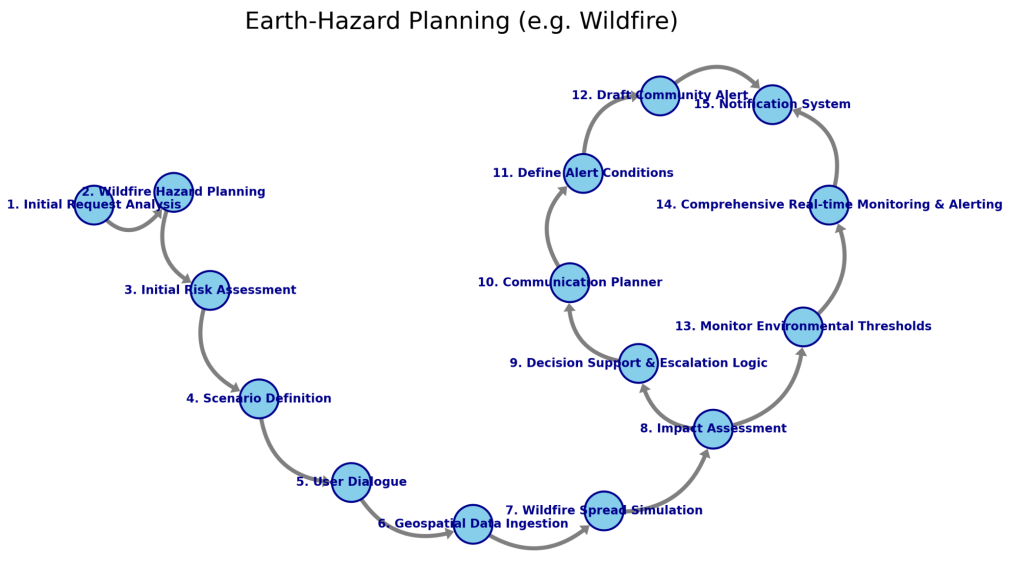

Earth-Hazard Planning (e.g. Wildfire)
1. Initial Request Analysis
2. Wildfire Hazard Planning
3. Initial Risk Assessment
4. Scenario Definition
5. User Dialogue
6. Geospatial Data Ingestion
7. Wildfire Spread Simulation
8. Impact Assessment
9. Decision Support & Escalation Logic
10. Communication Planner
11. Define Alert Conditions
12. Draft Community Alert
13. Monitor Environmental Thresholds
14. Comprehensive Real-time Monitoring & Alerting
15. Notification System

Figure 5: Example tool workflow for the aspirational target capability Earth-Hazard Planning.

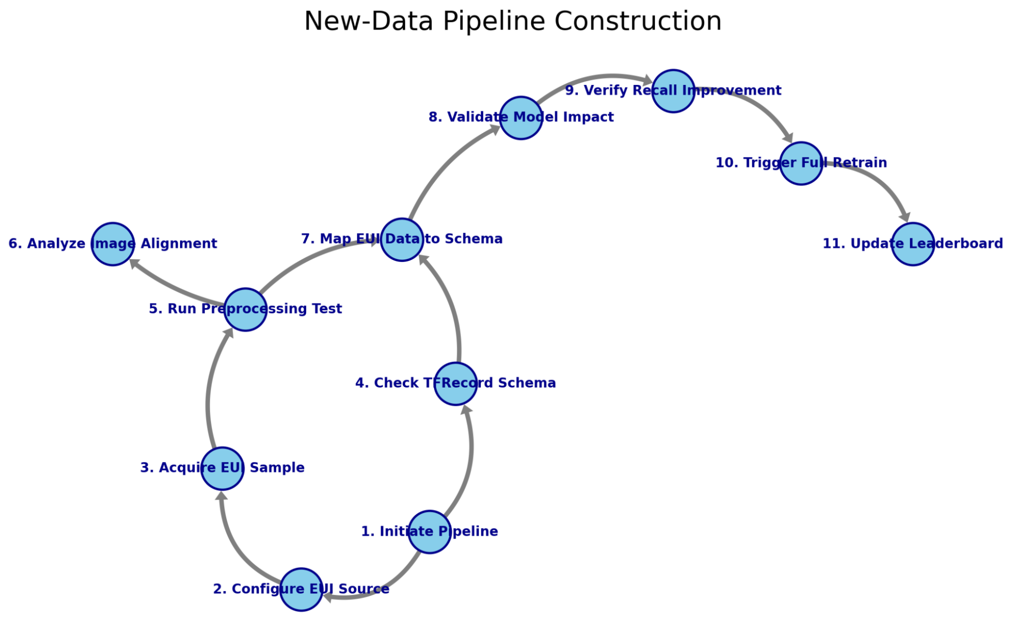


Figure 6: Example tool workflow for the aspirational target capability New Data Pipeline Construction.

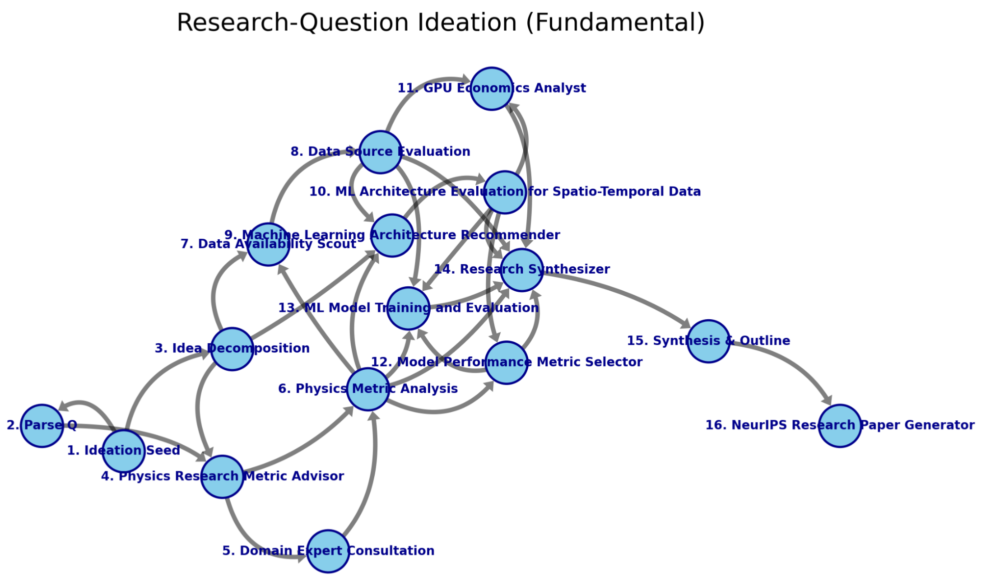

Research-Question Ideation (Fundamental)
1. Ideation Seed
2. Parse Q
3. Idea Decomposition
4. Physics Research Metric Advisor
5. Domain Expert Consultation
6. Physics Metric Analysis
7. Data Availability Scout
8. Data Source Evaluation
9. Machine Learning Architecture Recommender
10. ML Architecture Evaluation for Spatio-Temporal Data
11. GPU Economics Analyst
12. Model Performance Metric Selector
13. ML Model Training and Evaluation
14. Research Synthesizer
15. Synthesis & Outline
16. NeurIPS Research Paper Generator

Figure 7: Example tool workflow for the aspirational target capability Research Question Ideation (Fundamental).

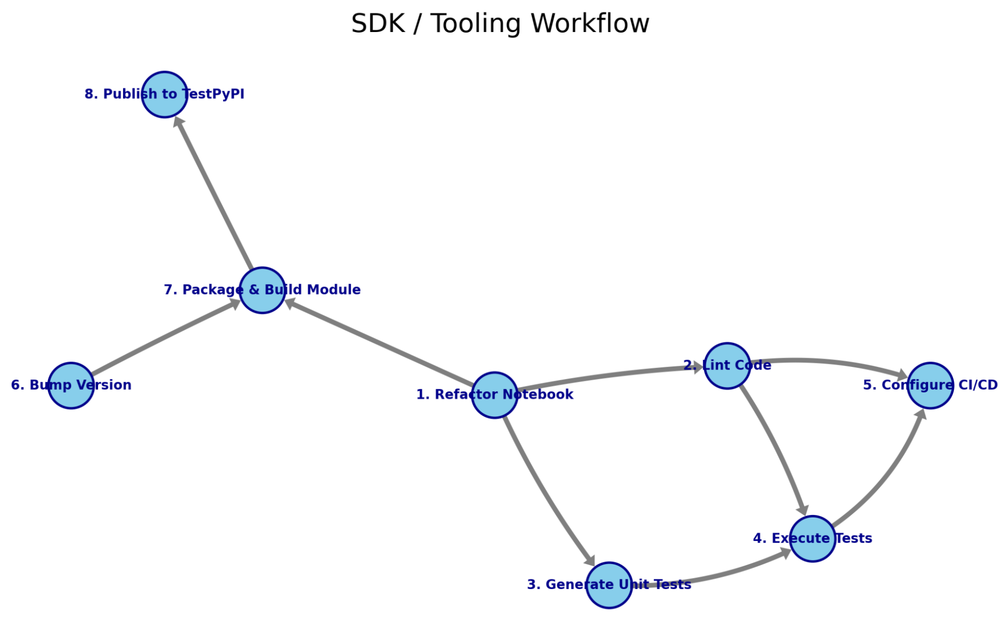


Figure 8: Example tool workflow for the aspirational target capability SDK / Tooling Workflow.

Table 3: Groups of tools generated from the aspirational target capabilities.

Group	Number of Tools	Description
research-analysis	14	*Scientific research, evaluation, consultation*
software-dev	13	*Code development, testing, integration*
emergency-mgmt	11	*Wildfire management, emergency planning*
machine-learning	11	*Model training, ML architectures, performance*
planning-config	11	*System configuration, planning horizons*
publishing-reporting	11	*Results publication, visualization, reporting*
data-ingestion	10	*Data acquisition from various sources*
data-processing	10	*Data preprocessing, schema handling*
decision-support	7	*Decision logic, escalation processes*
space-weather	6	*Solar events, space weather prediction*
monitoring-alerting	6	*Real-time monitoring, alerting systems*
satellite-ops	5	*Satellite command/control, safe-mode operations*
system-ops	4	*System operations, pipeline management*

Table 4: Demonstrated capabilities of the agentic orchestration system through working case studies.

Case Study	User Query	System Response
Ionospheric Modeling	*Generate vertical TEC for March 21, 2024*	Calls PyIRI MCP server, returns TEC profile visualization
	Show electron density at 40°N, 75°W	Routes to ionosphere agent, generates density vs. altitude plot
	Compare day/night ionosphere profiles	Executes parallel simulations, returns comparative analysis
Solar Surface Analysis	*Simulate active region evolution*	Routes to solar modeling agent, runs ARCADE simulation
	Predict sunspot growth over 24h	Calls ML model, returns growth probability map
	Identify emerging flux regions	Analyzes magnetogram data, highlights candidate regions
Data Pipeline Generation	*Create SDO/AIA download pipeline*	Generates Python script with appropriate data queries
	Design infrastructure for ML training	Creates Docker configuration and resource allocation plan
	Request approval for compute resources	Presents infrastructure diagram for human review
	Monitor pipeline execution	Provides real-time logging and progress updates
Tool Generation & Discovery	*What tools do I need for CME tracking?*	Analyzes requirements, suggests coronagraph processing tools
	Generate execution graph for flare prediction	Creates workflow diagram with required MCP tools
	List available heliophysics models	Returns catalog of integrated simulation and ML models

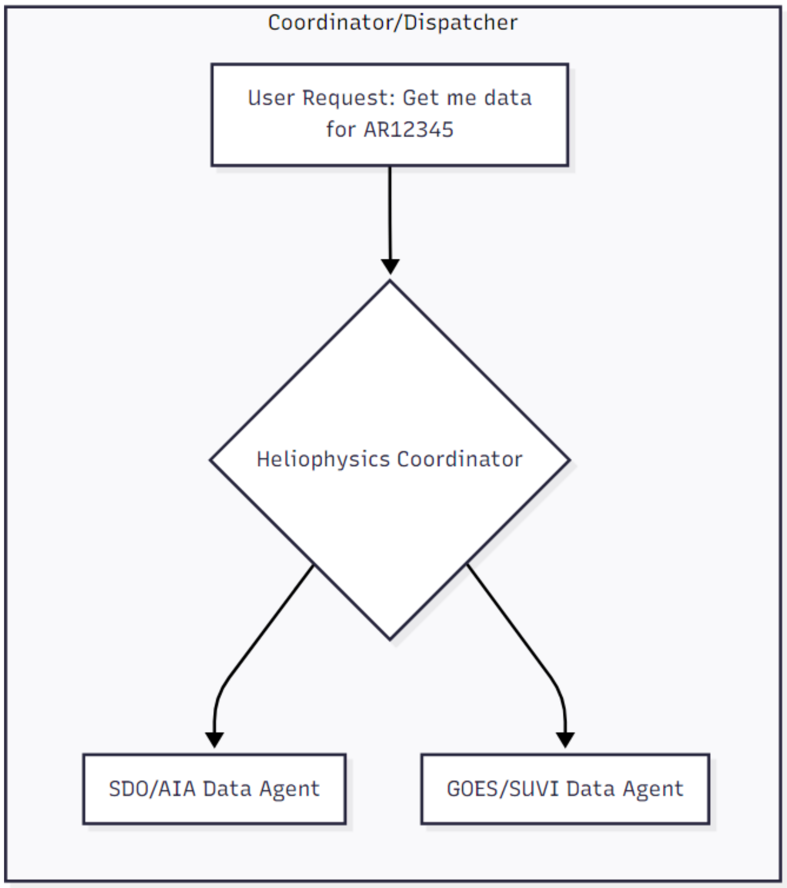


Figure 9: Depiction of the Coordinator/Dispatcher agentic design pattern applied to heliophysics. The user requests data from the coordinator, and then the coordinator breaks down the task of collecting `AR12345` from the SDO/AIA agent and the GOES/SUVI data agent.

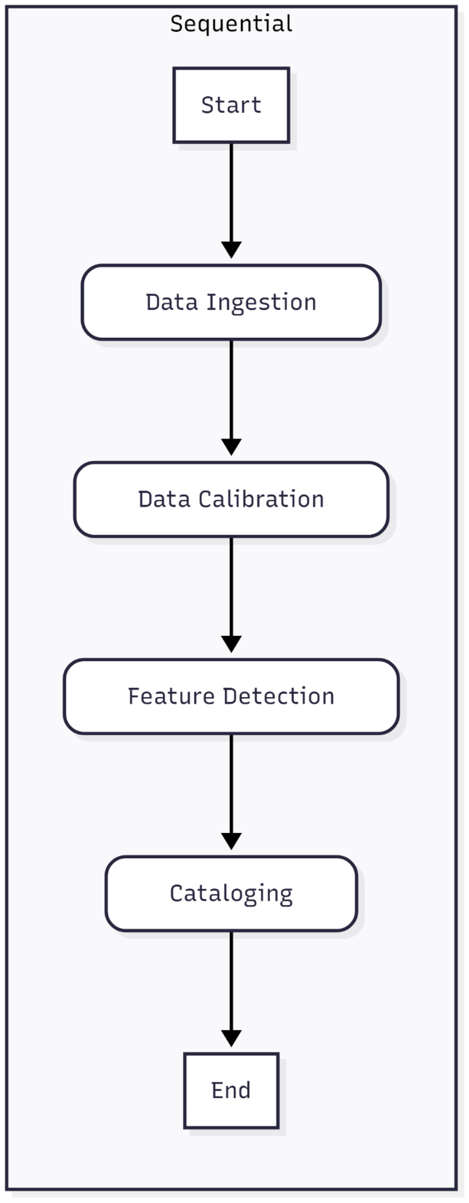


Figure 10: Depiction of the Sequential agentic design pattern applied to heliophysics. Actions are performed in order: downloading data, calibrating the downloaded data, detecting features in the data, and cataloging results.

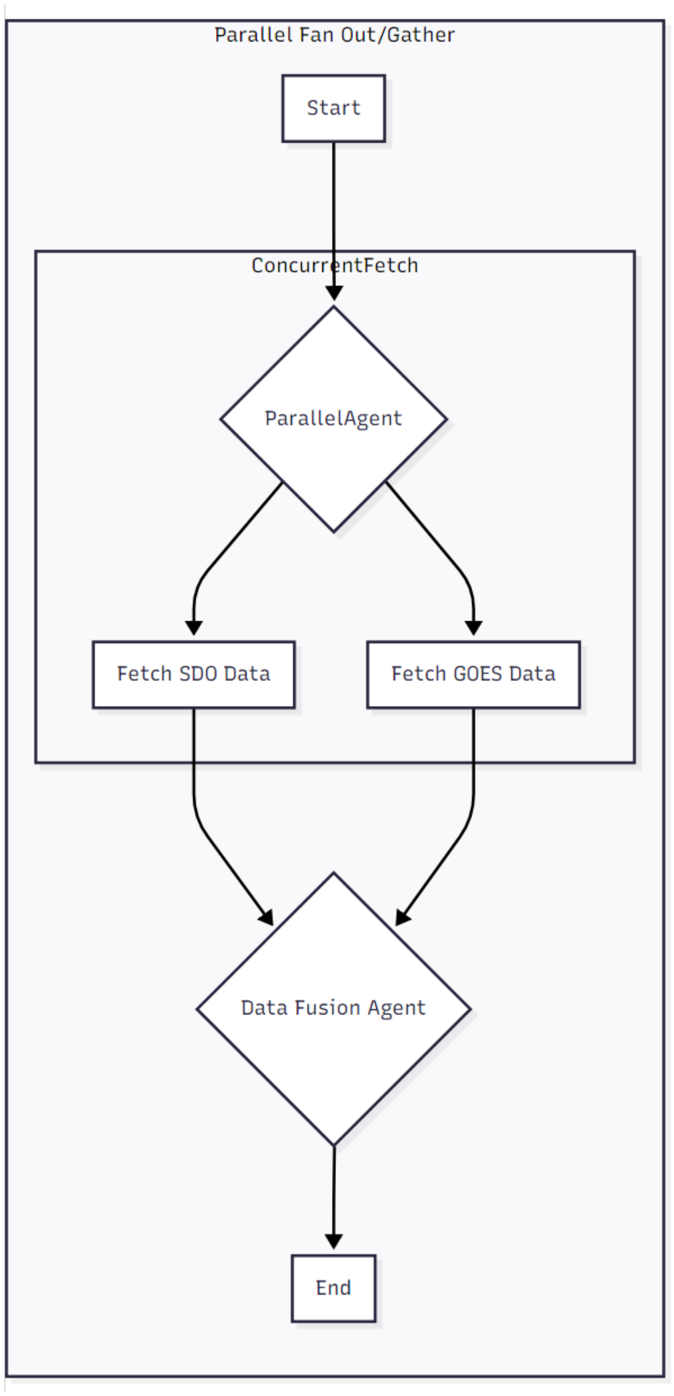


Figure 11: Depiction of the Parallel Fan Out/Gather agentic design pattern applied to heliophysics. The agents fetch SDO data and GOES data in parallel, and when those processes finish another agent puts the data together.

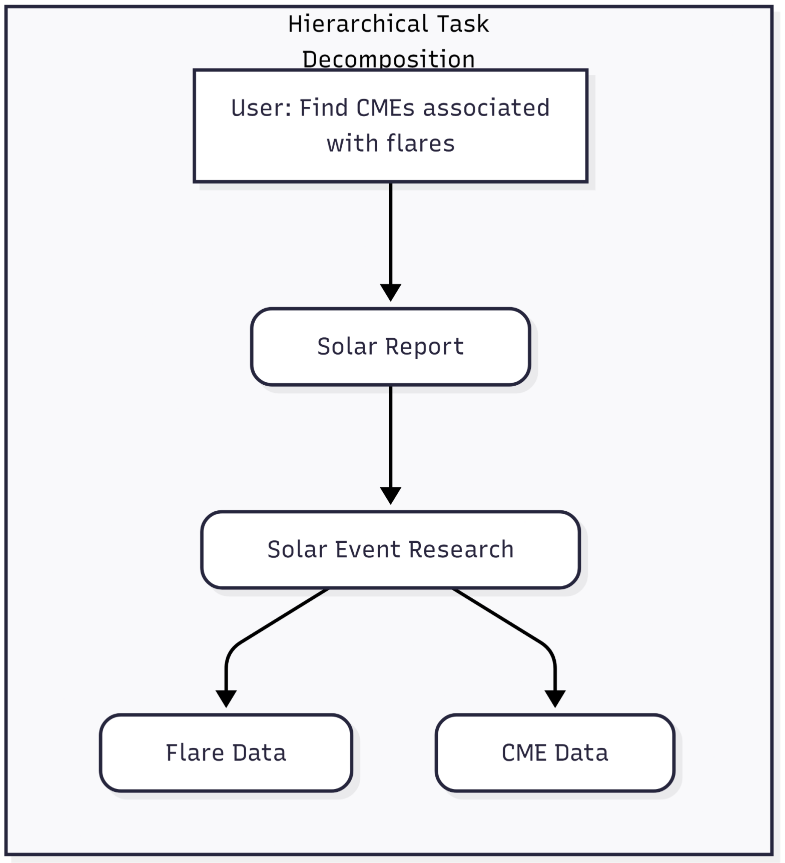

Hierarchical Task Decomposition
User: Find CMEs associated with flares
Solar Report
Solar Event Research
Flare Data
CME Data

Figure 12: Depiction of the Hierarchical Task Decomposition agentic design pattern applied to heliophysics. Tasks are broken down for collecting data from CMEs and flares, which will be used for however the user defines the Solar Event Research block.

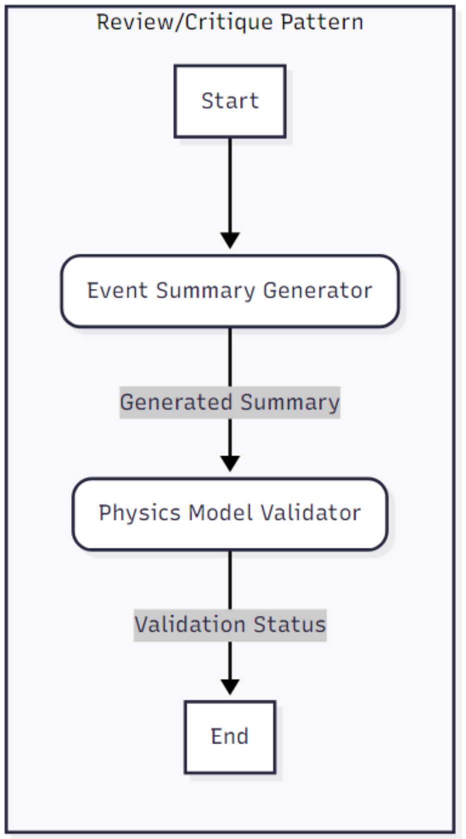


Figure 13: Depiction of the Review/Critique agentic design pattern applied to heliophysics. Events are summarized, and then a validator confirms the contents of the summary.

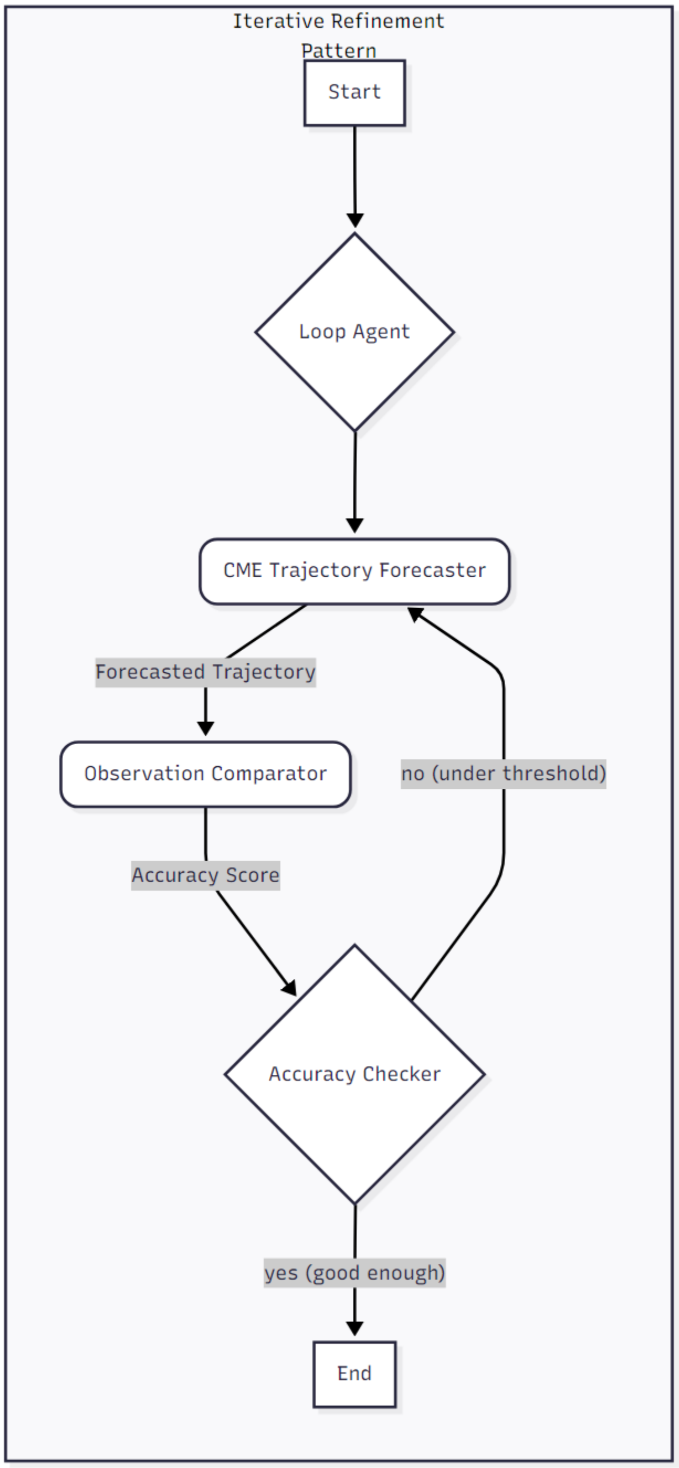


Figure 14: Depiction of the Iterative Refinement agentic design pattern applied to heliophysics. A CME forecaster is being built, new observations are added to the training data, and then the accuracy checker runs inference the test data to get an accuracy score. If the score does not meet the required threshold, more training data is added and the process repeats in a loop until the accuracy score is satisfactory.

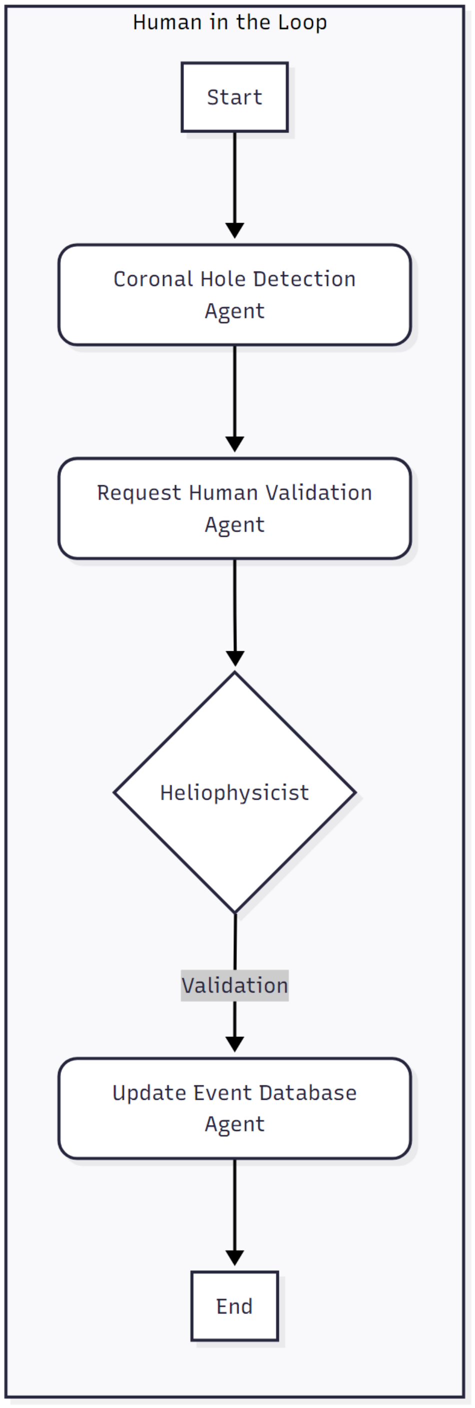
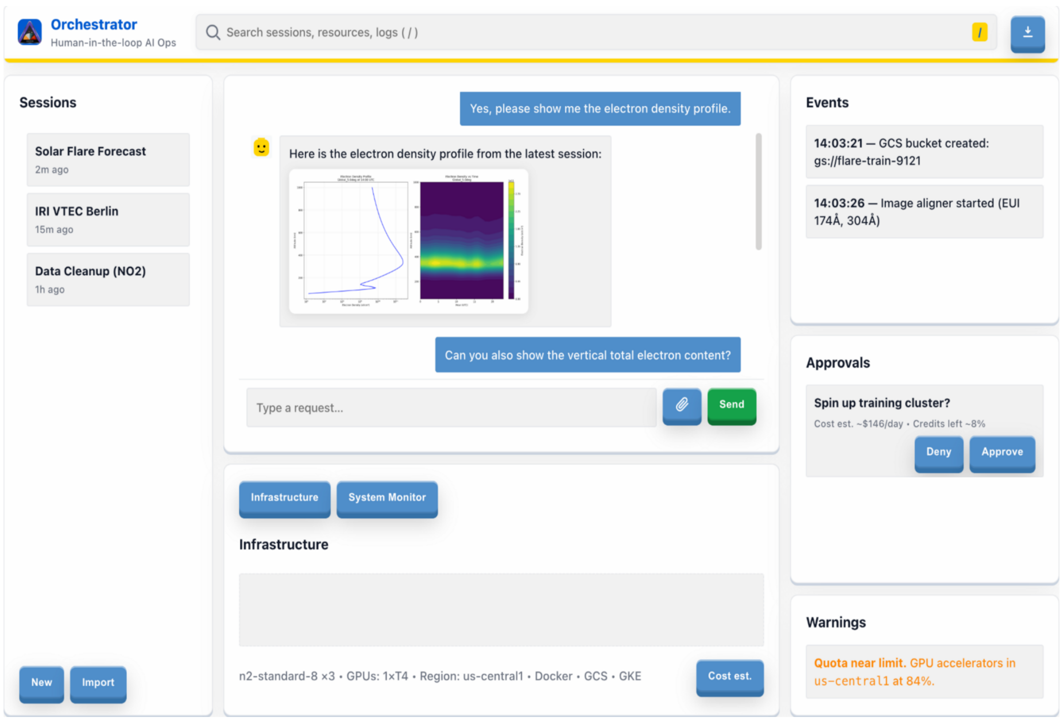

Human in the Loop
Start
Coronal Hole Detection Agent
Request Human Validation Agent
Heliophysicist
Validation
Update Event Database Agent
End

Figure 15: Depiction of the Human in the Loop agentic design pattern applied to heliophysics. An agent detecting coronal holes must ask a human heliophysicist to validate whether or not an image contains a coronal hole. The human's response is updated into the database.



Solar Flare Forecast
2m ago

IRI VTEC Berlin
15m ago

Data Cleanup (NO2)
1h ago

Yes, please show me the electron density profile.

Here is the electron density profile from the latest session:

Can you also show the vertical total electron content?

Infrastructure

n2-standard-8 ×3 • GPUs: 1×T4 • Region: us-central1 • Docker • GCS • GKE

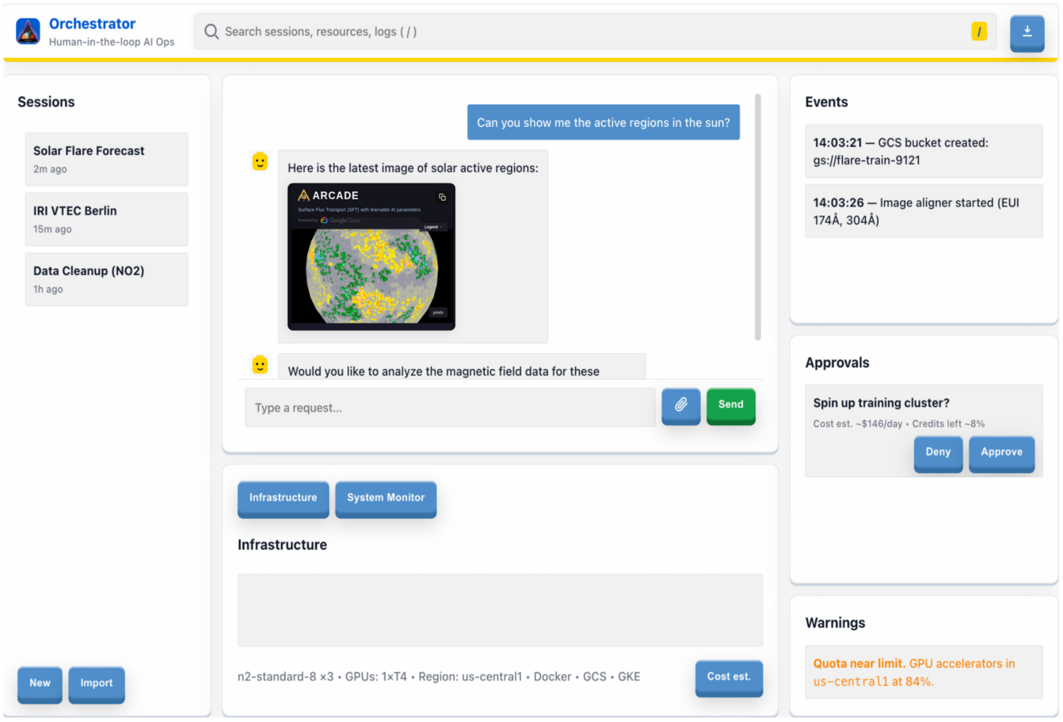
14:03:21 — GCS bucket created: gs://flare-train-9121

14:03:26 — Image aligner started (EUI 174Å, 304Å)

Spin up training cluster?
Cost est. ~$146/day • Credits left ~8%

Quota near limit. GPU accelerators in `us-central1` at 84%.

(a) Simulation outputs from International Reference Ionosphere model.



Solar Flare Forecast
2m ago

IRI VTEC Berlin
15m ago

Data Cleanup (NO2)
1h ago

Can you show me the active regions in the sun?

Here is the latest image of solar active regions:

Would you like to analyze the magnetic field data for these

Infrastructure

n2-standard-8 ×3 • GPUs: 1×T4 • Region: us-central1 • Docker • GCS • GKE

14:03:21 — GCS bucket created: gs://flare-train-9121

14:03:26 — Image aligner started (EUI 174Å, 304Å)

Spin up training cluster?
Cost est. ~$146/day • Credits left ~8%

Quota near limit. GPU accelerators in `us-central1` at 84%.

(b) Simulation outputs from active regions on the Sun's surface.

Figure 16: Simulation outputs from the agentic orchestration system for heliophysics from natural language queries for 16a International Reference Ionosphere model, and 16b active regions on the Sun's surface.

Table 5: Unit test examples generated by the agentic orchestration system for some of the case studies, and results and comments from the validator while running those tests.

Case Study	Unit Test	Test Result	Comment
Ionospheric Modeling	`assert (os.path.exists( vtec_file_path ), f"vTEC file not found: {vtec_file_path}")`	Pass	vTEC file exists passed!
	`assert ( os.path.getsize( vtec_file_path ) > 0, f"vTEC file is empty: {vtec_file_path}" )`	Pass	vTEC file size passed!
	`assert ( os.path.exists( profiles_file_path ), f"Profiles file not found: {profiles_file_path}" )`	Pass	Profiles file exists passed!
	`assert ( os.path.getsize( profiles_file_path ) > 0, f"Profiles file is empty: {profiles_file_path}" )`	Pass	Profiles file size passed!
Solar Surface Analysis	`assert ( response['status'] == 'success', f"Expected status 'success', but got {response['status']}" )`	Pass	status_check passed!
	`assert ( response['url'] is not None and response['url'] != '', "Expected a non-empty URL" )`	Pass	url_check passed!
	`assert ( response['data']['status'] == 'success', f"Expected data status 'success', but got {response['data']['status']}" )`	Pass	data_status_check passed!
	`assert ( response['parameters']['day'] == 25, f"Expected day 25, but got {response['parameters']['day']} )" # This test is expected to fail based on the tool's output for parameters.`	Fail	Expected day 25, but got 1. The model's parameters do not match the requested date.
	`assert ( response['parameters']['month'] == 8, f"Expected month 8, but got {response['parameters']['month']}" ) # This test is expected to fail based on the tool's output for parameters.`	Fail	Expected month 8, but got 1. The model's parameters do not match the requested date.
	`assert ( response['parameters']['year'] == 2025, f"Expected year 2025, but got {response['parameters']['year']}" ) # This test is expected to fail based on the tool's output for parameters.`	Fail	Expected year 2025, but got 2014. The model's parameters do not match the requested date.
	`assert response['parameters'] ['forecast_shift_hrs'] == 10, f"Expected forecast_shift_hrs 10, but got {response['parameters'] ['forecast_shift_hrs']}" )`	Pass	forecast_shift_hrs_check passed!