ES52 Internship Summary

Parker Melling (NASA Marshall Space Flight Center)

Introduction

- ES52 Intern
- Mentor: Hannah Hopkins
- Supervisor: Edward Camacho-Padilla
- Originally from Indiana
- Go to school at Georgia Tech
- Majoring in computer science and aerospace engineering
- Started at NASA in late January

Projects

- ARTEMIS Checksum Simulation Bug
- 1553 Validity Bits
- 1750a Validity
- Automatic Launch Sequence
- Attitude Hold
- Autogenerated Files
- Caution Warning Abort
- SPRAT to Sherlock Agent
- ARTEMIS Agent

How do checksums work with I-Loads?

- SLS Checksum Process:
    1. All relevant I-Load data is gathered (wind speed, atmospheric conditions, etc.)
    2. I-Loads with configured parameters are created.
    3. A checksum for the PAR data structure for when after the I-Load is applied is computed on the ground.
    4. The I-Load followed by the computed PAR checksum is sent from the ground to the rocket.
    5. The I-Load is validated by the FC (type, size, build number, struct alignment, etc.)
    6. The I-Load is applied to the PAR data structure.
    7. The PAR data structure is incrementally hashed via a hashing algorithm to compute the PAR checksum.
    8. The new checksum is compared against the checksum sent after the I-Loads to ensure that the entirety of the I-Load was transmitted and applied.

What is the problem?

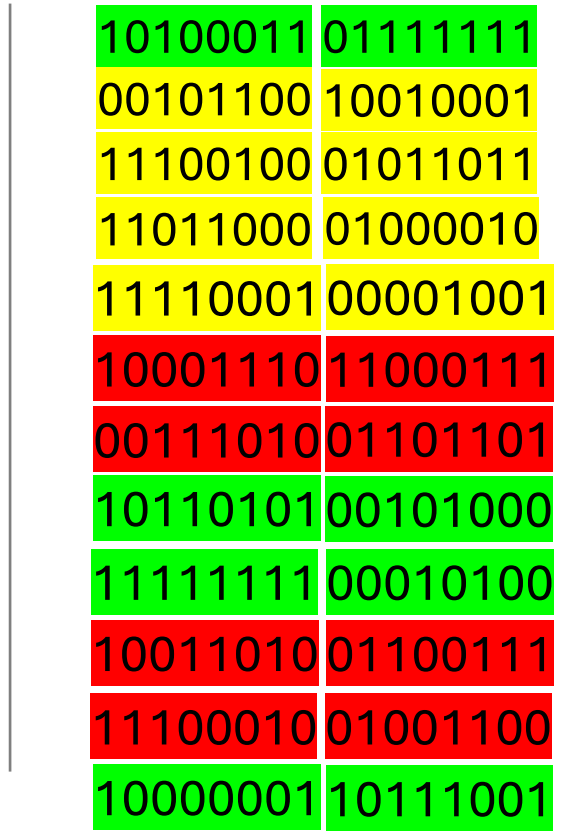
- In the ARTEMIS simulation, the emulated ground computer is of different endianness than the flight computer.
- The I-Load is correctly byte swapped and applied to the PAR.
- However, the checksums don’t match because the PARs are of different endianness and the hashing algorithm runs on raw bytes.
- This bug prohibits console support members, such as Lottie Ables (pictured right) from practicing checksum reporting before the day of launch.
- It also eliminates the ability to pinpoint the I-Load transmission as an issue in the ARTEMIS simulation when unexpected behavior occurs.

How I fixed the issue

- Currently, we are not calculating the checksums at the source and destination devices on the same endianness.
- Thus, we need to byte swap either the source or destination PAR.
- I chose to byte swap the PAR at the destination.
- However, byte swapping the PAR directly is not an option because the flight software utilizes the values in the PAR, which would be garbage if converted to the opposite endianness.
- Therefore, an auxiliary PAR data structure is necessary to hold the byte-swapped version of the PAR, which can then be passed into a hashing algorithm to compute the PAR checksum.

How I fixed the issue (cont.)

- Within flight software, there exist large structures to encapsulate similar data.
- In the ARTEMIS repository, a file exists that contains an array of structures.
- Each structure provides information about a singular piece of data, such as its pointer offset, multiplicity, datatype, and which large structure it belongs to.
- By iterating through this array of structures and utilizing the information, a byte swapped version of the PAR can be created.



Impact

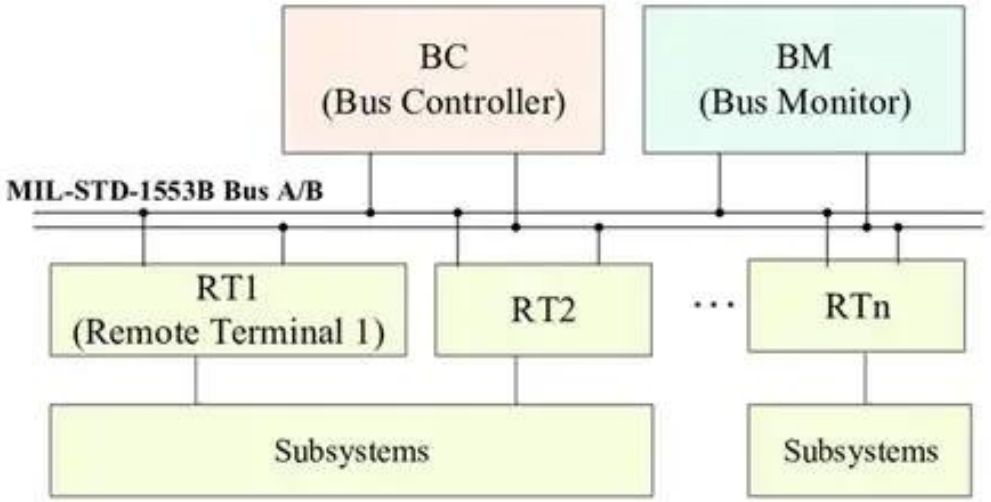
- Individuals on console, such as Lottie Ables (pictured right), can now correctly practice their role in validating the PAR checksum before the day of launch.
- The ARTEMIS simulation now has the capability to pinpoint I-Load data transmission as the issue when unexpected behavior occurs.

What is Sherlock

- Sherlock is a Python-based data analysis tool for ARTEMIS simulation data.
- It enables developers to make changes to flight software code, run a simulation, and quickly check that their changes have their intended effects without breaking anything else.

Sherlock 1553 Validity Bits Module

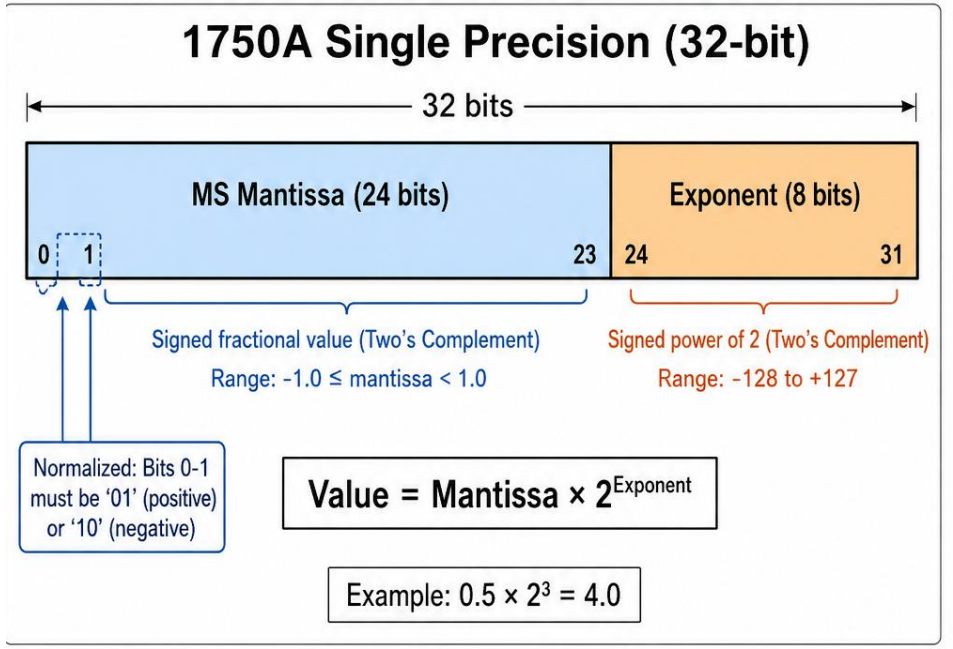
The MIL-STD-1553B is a reliable bus architecture utilized for SLS.

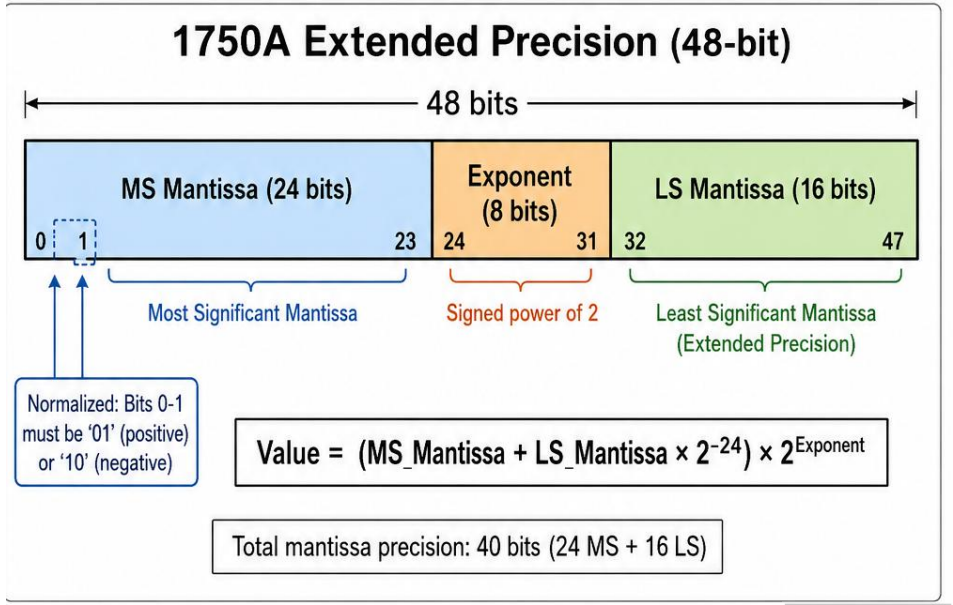


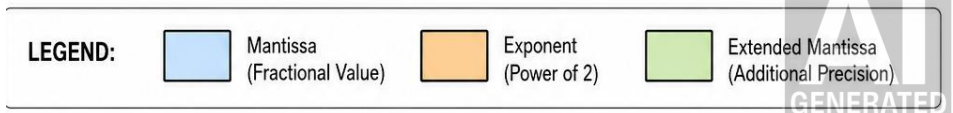
Artemis II Launch Experience

What is MIL-STD-1750a

- MIL-STD-1750a is a military standard to represent floating point numbers.
- The bits of a 1750a number are partitioned into a mantissa and an exponent.
- The mantissa utilizes two's compliment to represent fractional numbers from -1.0 to 1.0, which is then multiplied by 2 raised to the exponent.
- $value = mantissa \times 2^{exponent}$
- The mantissa must be normalized ($|mantissa| \geq 0.5$), so every floating point number has a unique representation (why only 01 or 10 is valid).







Sherlock 1750a Validity Module

- I developed a module in Sherlock to validate all 1750a CUIs:
  1. Queries IMACS database to find all CUI definitions that hold 1750a floating point numbers.
  2. Creates message filters for each definitions and finds all CUIs in the simulation that contained the 1750a numbers.
  3. Extract mantissa from the raw bytes of the message body.
  4. Checks both single and extended precision 1750a values for opposite sign and next mantissa bits.
  5. Reports the first occurrence of any invalid value for each CUI.
- I created a README to explain how the module works and how to use it.
- I also created a script with 13 comprehensive unit and integration tests.

Intern Q&As

Periodically, Hannah set up Q&As with various employees, so interns could gain a better idea of all the projects going on at NASA.

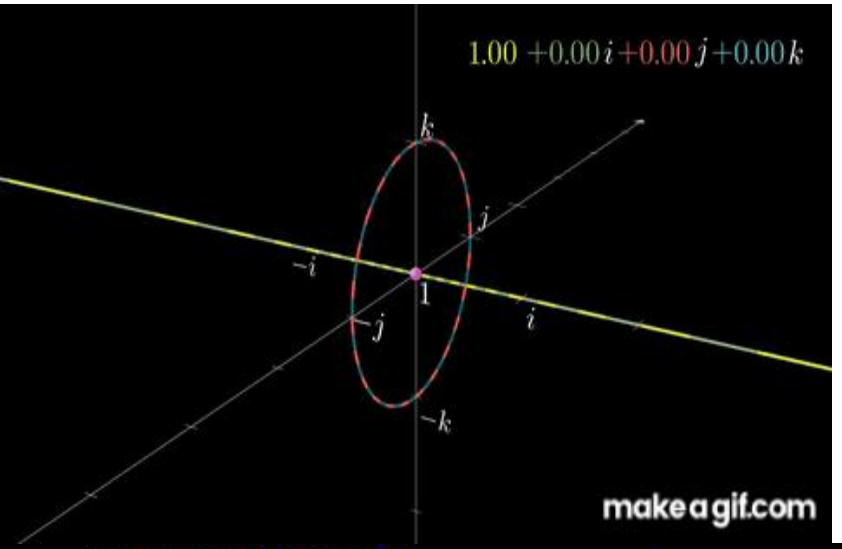
We had the opportunity to meet with with Jonathan Mack, Phillip Fox, Bob Hawkins, Anthony Maenza, Steven Hough, and Paul Doyle.

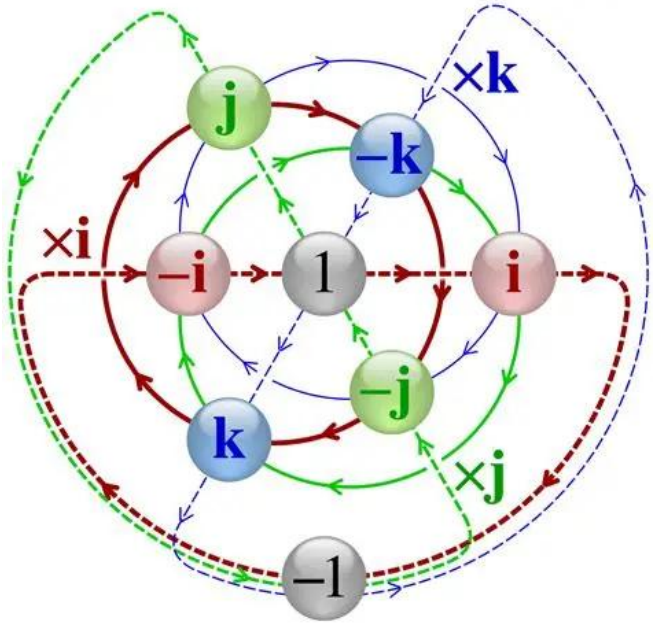
We learned about various topics from GN&C to how lightning affects launch operations.

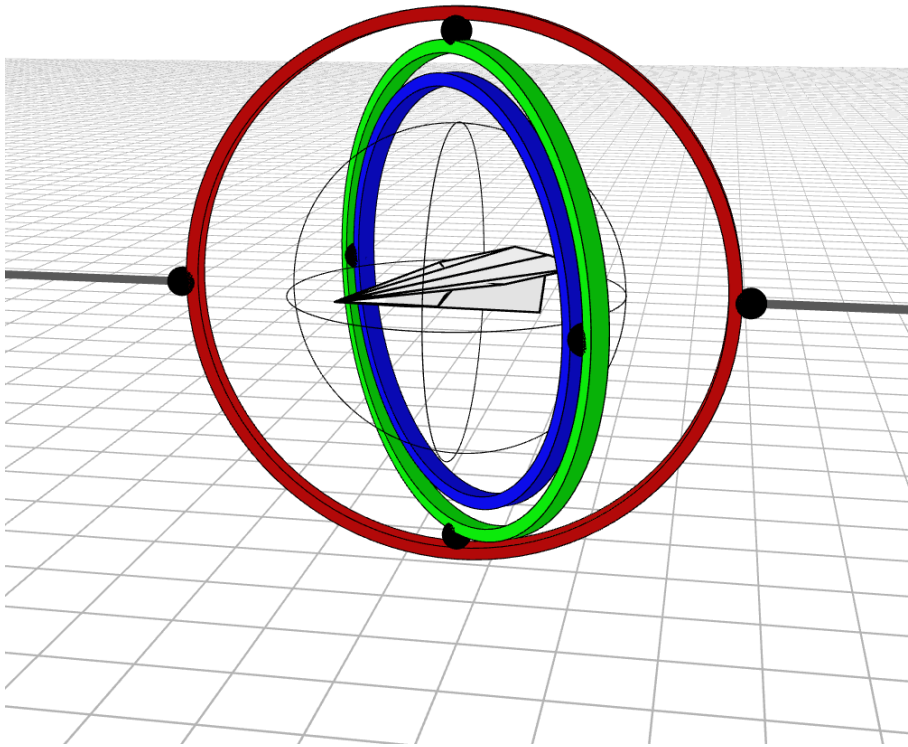
Sherlock ALS Module

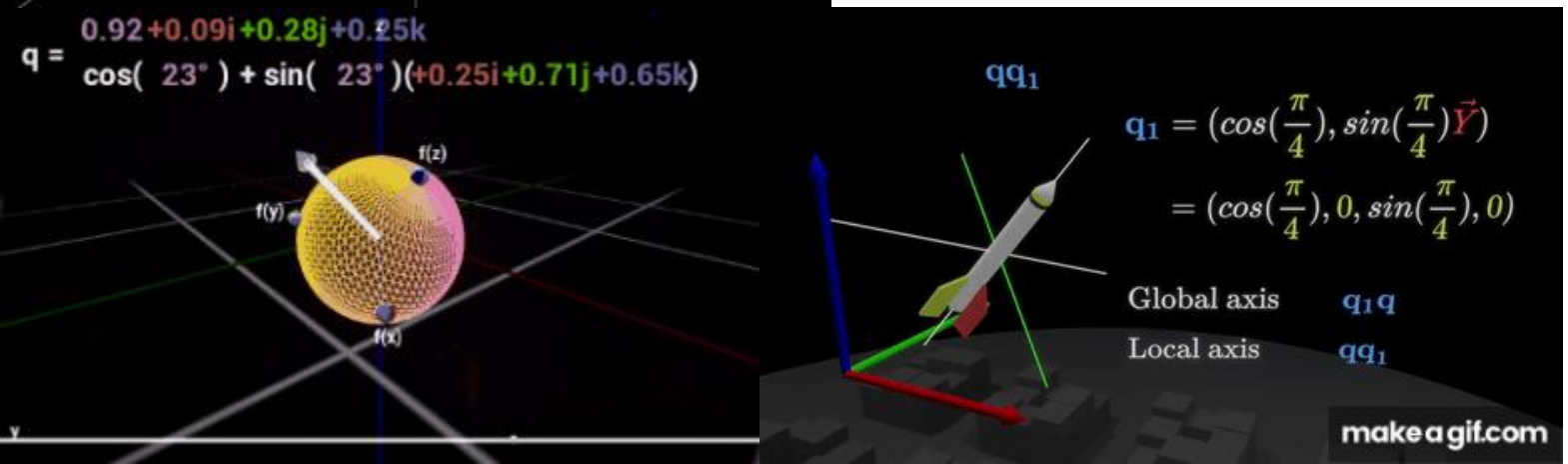
- I created a module in Sherlock to verify nominal ALS simulation launch behavior:
  1. Extracts commanded launch time from the GoForALS command.
  2. Verifies the FC acknowledges command within a short time frame.
  3. Validates FC enters ALS mode at the proper time.
  4. Ensures the rocket launched within one minor frame of the commanded launch time.
  5. Obtains the planned launch time from the modified PAR database in the simulation to verify the rocket was launched during the planned launch window.

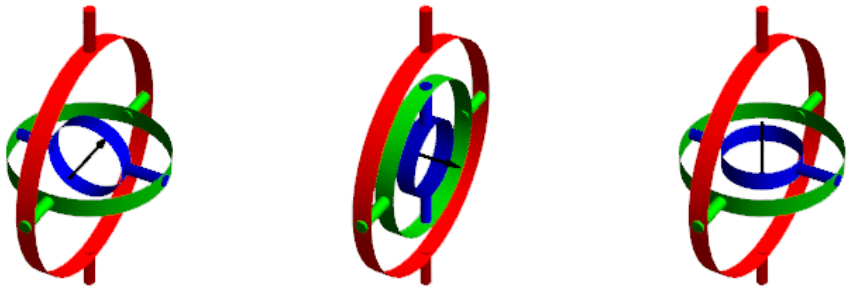
Quaternions











Sherlock Attitude Hold Module

I created an attitude hold module to ensure nominal attitude behavior from the rocket upon stage 3 separation in simulation:

1. The module ensures the booster separation commands were sent, the software operation mode transitioned to core stage flight, and the boosters acknowledged the command.
2. The PAR database is queried to obtain GNC's monitor duration.
3. At the start of stage 3, a reference quaternion is created.
4. Every minor frame is looped through, and the current quaternion is obtained to record attitude drift over time.

Autogenerated Files Module

Many source codes files (telemetry, fault management, commands, GNC, parameters) are automatically generated by specialized build utilities instead of manually created.

Created Sherlock module to iterate over autogenerated files within FSW, utilizing regular expressions to extract version information from comments.

Compiles information into a clean report table for easy version verification.

Caution Warning Abort Module

During launch, FSW can send cautions, warnings, and abort recommendations if off-nominal behavior occurs.

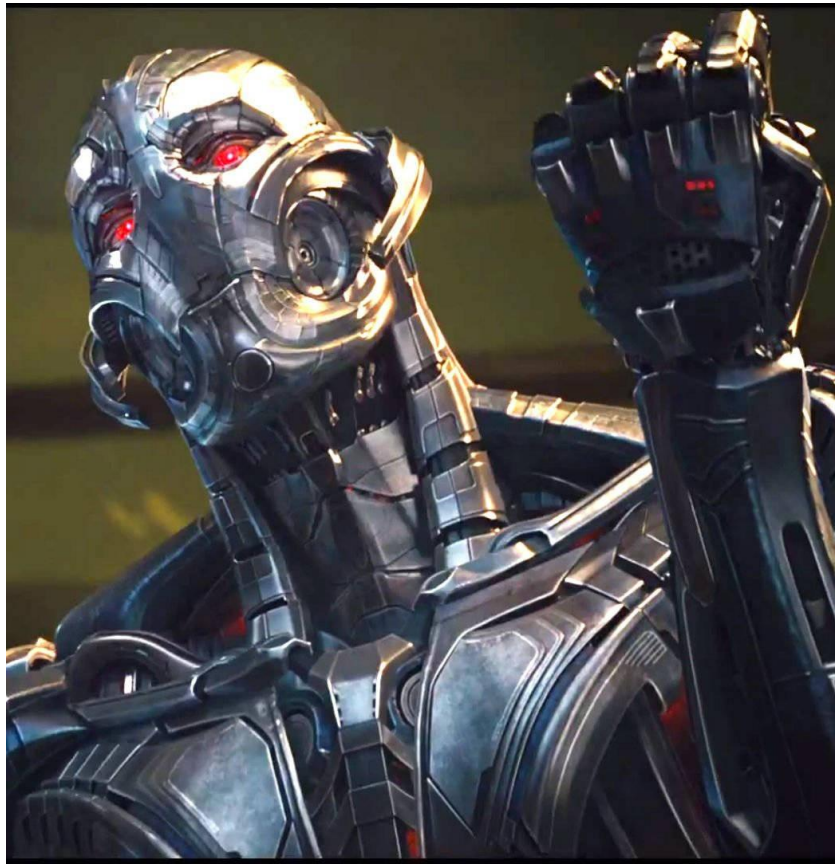
Some are expected, such as loss of communications with the MPCV after main engine cutoff (MECO).

Created a module to check FSW data exchange messages (DEMs) to determine if any cautions, warnings, or abort recommendations have been sent.

Enables FSW developers to determine if their code changes triggered any cautions, warnings, or abort recommendations.

Sprat to Sherlock Agent

- I realized that the process of creating new Sherlock modules could be greatly sped up with the utilization of AI.
- I created an agent called *Sherlock Full Module Creator*, which can greatly speed up the module creation process.
- I created a knowledge package that the agent has access to, which includes Sherlock documentation, an example module, CUIS/DEMS/PARS, and additional knowledge not included in the Sherlock documentation.
- The agent is composed of seven subagents:
- Sherlock Full Module Creator → SPRAT Expert → PAR/CUI/DEM Expert → Sherlock API Expert → Integration Expert → Test Expert → README Expert
- Disclaimer: human oversight is still absolutely necessary



ARTEMIS Agent

- Learning ARTEMIS is very arduous and lengthy.
  - Like drinking from a firehose
- I read a 333-page “ARTEMIS book” before developing any code.
- Created an ARTEMIS agent capable of answering most questions for easier onboarding and later reference.
- Cites sources for verification and further reading.

Mission Accomplished - Statistics

- 7,141 lines of code written
- 7 pull requests
- 100+ SSH Sessions
- 1 rocket launch witnessed